S802 用户手册

Number	
Revision	1.2.15
Security	Public
Date	2017-07-28

平头哥半导体有限公司 T-HEAD Semiconductor Co.,LTD

地址: 杭州市西湖区西斗门路 3 号天堂软件园 A 座 15 楼 邮编: 310012

网址: www.c-sky.com

版本历史:

版本	日期	描述	作者
1.0	12/01/2011	1. 第一版基本功能描述。	平头哥半导体有限公司
1.1	12/15/2011	1. 删除第七章中对 DIV 指令的描述。 2. 修改 8.2.2 未对齐异常描述，修改为只能发生在数据上。 3. 删除 8.2.7 的跟踪异常，S802 没有设计跟踪模式，即不会产生异常。 4. 删除第 8 章中除以 0 异常中的内容，S802 不实现除法指令。	平头哥半导体有限公司
1.2	2/13/2012	1. 修改对 32/16 的指令描述。 2. 修改对通用寄存器的描述。 3. 增加二进制代码转译功能的描述。	平头哥半导体有限公司
1.2.1	3/1/2012	1. 修改 S802 系统结构图。 2. 修改 S802 信号框图。	平头哥半导体有限公司
1.2.2	3/8/2012	1. 修改表达错误。	平头哥半导体有限公司
1.2.3	1/13/2014	1. 增加 BPUSH.H、BPUSH.W、BPOP.H、BPOP.W。 2. 更新 MGU 相关控制位功能。 3. 增加 SEU 单元。 4. 增加部分 32 位指令。	平头哥半导体有限公司
1.2.4	2/16/2014	1. 修正 CR21 名称表示错误 2. 修正总线结构 prot 信号表征含义 3. 加入 strong order 机制说明	平头哥半导体有限公司
1.2.5	3/28/2014	1. 加入软件复位控制寄存器以及软件复位机制说明	平头哥半导体有限公司
1.2.6	5/14/2014	1. 修改 MGU 配置寄存器相关说明	平头哥半导体有限公司
1.2.7	6/30/2014	1. 修改 lrw16 编码，删除 bsr16，增加 btsti16	平头哥半导体有限公司

1.2.8	9/3/2014	1. 删除部分 32 位指令 2. 删除 LDR 指令 3. 对异常处理机制进行补充说明 4. 对调试机制进行修改 5. 加入 cache 相关说明 6. 加入 MM 位对非对齐异常的控制说明	平头哥半导体有限公司
1.2.9	11/13/2015	1. 修改 MGU 相关寄存器说明	平头哥半导体有限公司
1.2.10	7/14/2016	1. 去除软件复位寄存器	平头哥半导体有限公司
1.2.11	8/6/2016	1. 增加指令说明	平头哥半导体有限公司
1.2.12	8/29/2016	1. 增加软件复位说明 2. 增加中断响应加速使能说明	平头哥半导体有限公司
1.2.13	12/08/2016	1. 部分勘误	平头哥半导体有限公司
1.2.14	4/5/2017	1. 调整排版，和 803S 一致 2. 删除数据总线描述 3. 增加 Cache、总线等描述，和 803S 一致 4. 删除程序计数器相关描述（PC），该寄存器是无法被操作的，无需列出来 5. 增加中断 SP 相关描述 6. 增加 grs 指令相关内容 7. 增加栈保护相关内容 8. 增加地址监测异常内容 9. 增加 TS pending 10. 其他琐碎修改点	平头哥半导体有限公司
1.2.15	7/28/2017	1. 调整 MGU 相关内容	平头哥半导体有限公司

目录：

图表目录：

1. 概述

1.1. 简介

S802 是平头哥半导体有限公司自主研发的极低功耗、极低成本嵌入式 CPU 核，以 8 位 CPU 的成本获得 32 位嵌入式 CPU 的运行效率与性能。S802 基于 C-SKY V2 自主指令架构，采用 16/32 位混合编码系统，通过精心设计指令系统与流水线硬件结构，具备极低成本、极低功耗和高代码密度等优点。S802 主要针对智能卡、智能电网、低成本微控制器、无线传感网络等嵌入式应用。

S802 采用了 16/32 位混合编码的 RISC 指令集，实现了 C-SKY V2 指令架构中 65 条 16 位指令和部分 32 位指令。其中 16 位指令集的优势是低成本、高代码密度，缺点是索引和立即数范围较小；32 位指令集的优势是立即数和相对跳转偏移量宽、操作数多、性能强。在实际使用中，C-SKY 编译器会根据编译优化的实际需求，有选择的选用 16 位和 32 位指令混合。用户在使用汇编时，仅需要按照需求书写统一格式的汇编指令，汇编器会根据实际情况选择 16 位或者 32 位指令，指令宽度对用户透明。

1.2. 特点

S802 处理器体系结构的主要特点如下：

- RISC 精简指令结构；
- 32 位数据，16/32 位混合编码指令；
- 2 级顺序执行流水线；
- 可配置的硬件乘法器，支持 1 个周期快速产生乘法结果；
- 单周期指令和数据存储器访问；
- 无延时的分支跳转；
- 支持 AHB-Lite 总线协议，支持可配置的指令总线；
- 支持多种处理器时钟与系统时钟比；
- 支持大端和小端；
- 支持可配置内存保护区域（0-8）；
- 支持可配置安全扩展单元；
- 支持可配置可信防护技术；
- 支持可配置紧耦合 IP，包括系统计时器，矢量中断控制器等；
- 支持可配置的二进制代码转译机制；
- 支持可配置的高速缓存器，高速缓存容量 2KB、4KB 和 8KB 硬件可配。

S802 在 SMIC 55nm 工艺下性能参数如下：

- 工作频率 50MHz（最恶劣情况）；
- CPU 基本核面积约 12.5K 等效门；
- 动态功耗小于 11 uW/MHz；
- 性能：0.95~1.3DMIPS/MHz。

1.3. 可配置选项

S802 可配置选项如下表所示

图表 1-1 S802 可配置选项

可配置单元	配置选项	详细
硬件乘法器	无 /有	若配置则 1 个周期产生乘法结果，否则要 3-34 周期完成。
内存保护单元	0 到 8 个表项	可以配置为 0-8 个表项，其中 0 表示不实现内存保护单元。
高速缓存器	无 /2K /4K /8K	可以配置为 2KB、4KB、8KB。
可信防护技术	无 /有	配置该技术，结合中天微公司的 SoC 平台技术/系统软件，将提供系统的安全防护功能。
安全抗攻击技术	无 /有	配置该技术，将提供针对非侵入式/侵入式硬件攻击的防护
指令总线	无 /Flop-out /Non-Flop-out	在配置了指令总线的情况下，又支持寄存器输出（Flop-out）和直接输出（Non-Flop-out）两种方式。
系统总线	兼容 AHB/ 兼容 AHB Lite	可以配置为兼容 AHB 协议或者兼容 AHB Lite 协议。
矢量中断控制器	无 /INT16 /INT32	支持硬件中断的嵌套处理。支持 16 个中断源、32 个中断源。
系统计时器	无/有	用于计时。

1.4. 可测性设计

S802 支持扫描链测试（SCAN）和内建自测试(BIST)。其中，扫描链测试用于测试处理器内部的组合和时序逻辑是否存在制造错误，内建自测试用于测试高速缓存是否存在制造错误。

S802 的扫描链数目可由客户指定。

1.5. 可调式性设计

S802 使用 JTAG 标准（2 线）设计硬件调试接口。S802 支持所有常见的调试功能，包括软断点、内存断点，改寄存器检查和修、存储器检查和修改，指令单步跟踪与多步跟踪、程序流跟踪等。具体请详见第九章——调试接口。

1.6. 命名规则

1.6.1.符号

本文档用到的标准符号和操作符如下表所示

符号	功能
+	加
-	减
*	乘
/	除
>	大于
<	小于
=	等于
≥	大于或等于
≤	小于或等于
!=	不等于
.	与
+	或
⊕	异或
NOT	取反
:	连接
⇒	传输
⇔	交换
±	误差
0b0011	二进制数
0x0F	十六进制数

1.6.2.术语

- 逻辑 1 是指对应于布尔逻辑真的电平值。
- 逻辑 0 是指对应于布尔逻辑伪的电平值。
- 置位是指使得某个或某几个位达到逻辑 1 对应的电平值。
- 清除是指使得某个或某几个位达到逻辑 0 对应的电平值。
- 保留位是为功能的扩展而预留的，没有特殊说明时其值为 0。

- 信号是指通过它的状态或状态间的转换来传递信息的电气值。
- 引脚是表示一种外部电气物理连接，同一个引脚可以连接多个信号。
- 使能是指使某个离散信号处在有效的状态：
  低电平有效信号从高电平切换到低电平；
  高电平有效信号从低电平切换到高电平。
- 禁止是指使某个处在使能状态的信号状态改变：
  低电平有效信号从低电平切换到高电平；
  高电平有效信号从高电平切换到低电平。
- LSB 代表最低有效字节,MSB 代表最高有效字节。

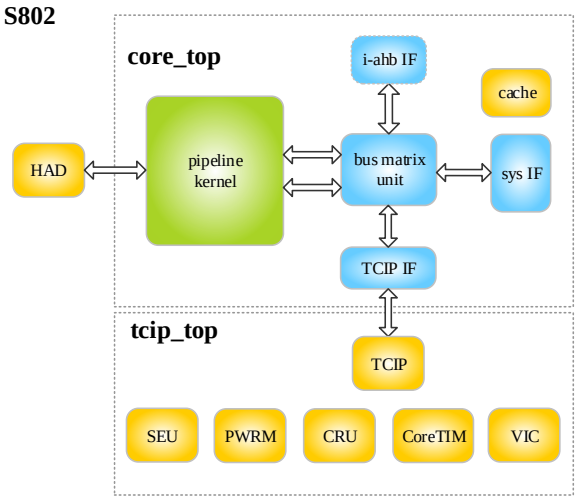
存储单元和寄存器当“pad_sysio_bigend_b=0”时采用大端模式，其字节次序是高字节在最低位。一个字中的字节是从最高有效字节(第 31—24 位)开始往下排列。

- 当“pad_sysio_bigend_b=1”时，采用小端模式。
- 信号，位域，控制位的表示都使用一种通用的规则。
- 标识符后来跟着表示范围的数字，从高位到低位表示一组信号，比如 addr[4:0]就表示一组地址总线，最高位是 addr[4]，最低位是 addr[0]。

单个的标识符就表示单个信号，例如 pad_cpu_rst_b 就表示单独的一个信号。有时候会在标识符后加上数字表示一定的意义，比如 addr15 就表示一组总线中的第 16 位。

2. 微体系结构

2.1. 结构框图



图表 2-1 S802 结构图

S802 处理器采用 2 级流水线结构。指令取指阶段主要负责从内存中获取指令，并对 16/32 位变长指令进行译码、复杂指令拆解和调度指令发射到下一级流水线；指令执行阶段主要负责指令的执行和结果的回写。S802 中内存数据的存取划分为两个步骤，分别为地址的产生和内存的访问，最快支持在一个时钟周期内完成存储器的访问。

可配置的内存管理单元支持超级用户自定义内存空间的访问权限，权限划分为：不可读写/只读/可读写，可执行/不可执行，也可以设置为安全区与非安全区。

总线接口单元兼容 AHB-Lite 协议，设计有寄存器输出（Flop-out）和直接输出（Non-Flop-out）两种硬件配置。在寄存器输出配置下，系统时钟与 CPU 时钟比例（1:1，1:2，1:3，1:4，1:5，1:6，1:7，1:8）下工作；在直接输出配置下，系统时钟与 CPU 时钟只能按照 1：1 工作。

硬件辅助调试单元支持各种调试方式，包括软件设置断点方式、内存断点方式、单步和多步指令跟踪等 7 种方式，可在线调试 CPU、通用寄存器（GPR）、协处理器 0（CP0）和内存。

片内外存储资源包括紧耦合存储器接口、片上紧耦合的 IP 接口和系统总线接口。紧耦合存储器接口用于用户自定义功能扩展，片上紧耦合的 IP 接口下设计有中断控制器（VIC）、系统计时器（CoreTIM）、功耗管理单元（PWRM）、高速缓存控制寄存器单元（CRU）和安全扩展单元（SEU）。矢量中断控制器支持 16/32 个中断源，支持电平和脉冲两种中断方式。系统计时器提供 1 个 24 位的循环递减计数器，计数器按照 CPU 时钟或者外部参考时钟递减计数，计数到 0 时产生中断请求。功耗管理模块支持动态功耗和静态功耗的模式控制。高速缓存控制寄存器单元在配置有高速缓存（cache）时负责对其进行控制及操作。S802 同时设计有针对信息安全应用的可配置模块。

S802 处理器实现了二进制代码转译机制，支持对 JAVA 等解释性语言的加速。用户可以通过选用支持该功能的 S802 核，实现对 JAVA 应用的加速。

*注：详细内容请参考《CSKY 处理器紧耦合 IP 用户手册》。

2.2. 流水线介绍

本章介绍关于 S802 的指令流水线和指令时序信息。

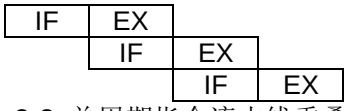
S802 微处理器有 2 级流水线： 即指令提取与译码、指令执行与退休。2 级流水线的作用如图表 2-2：

流水线名称	缩写	流水线作用
指令提取与译码	IF	1、访问指令总线； 2、计算下一条指令的地址； 3、分支地址计算； 4、指令预译码； 5、指令译码； 6、复杂指令拆解；
指令执行与退休	EX	1、 指令发射； 2、 访问寄存器组； 3、 指令的执行； 4、 LOAD/STORE 指令的数据地址的产生； 5、 访问数据总线； 6、 指令执行结果回写。

图表 2-2 各级流水线作用

在流水执行指令的过程中，采用单发射机制，即一个时钟周期至多发射一条指令；同时采用阻塞发射架构，即前一条指令没有执行完成时，后续指令不得发射到执行单元。

单周期指令流水线重叠执行顺序如图表 2-3 所示，LOAD, STORE、算术和逻辑指令都属于这类指令。

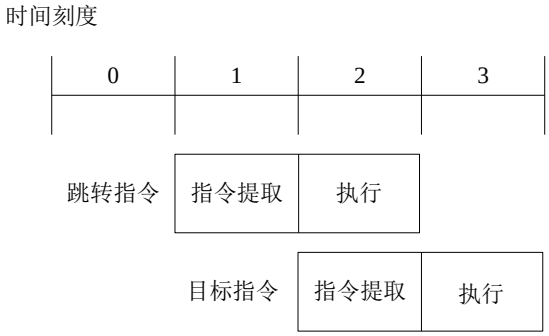


图表 2-3 单周期指令流水线重叠执行

乘法指令 MULT 需要多个执行周期完成（不可流水操作），如图表 2-4 所示：

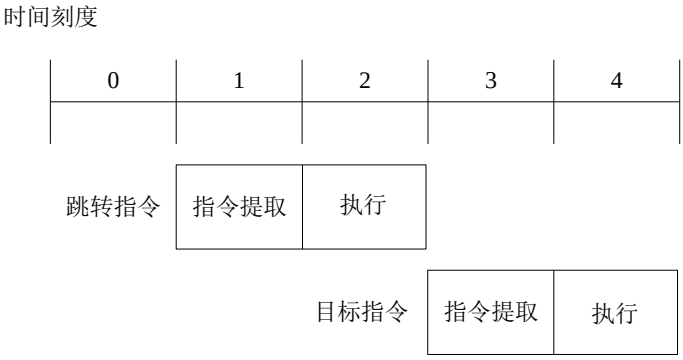
图表 2-4 乘法指令 mult 的执行过程

BR, BSR 指令的跳转和条件分支指令，由于采用了提前获取条件位技术，即使出现跳转流水线也不停顿，其执行过程如图表 2-5 所示；



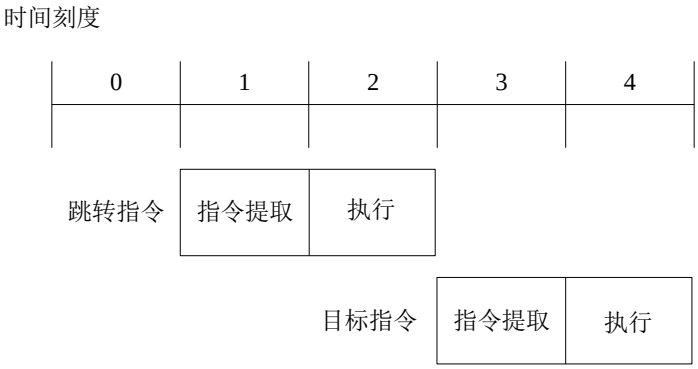
图表 2-5 BR, BSR 指令的跳转和条件指令预测正确时的执行过程

JMP 指令（JMP R15 指令除外）至少需要两个周期来填充流水线，其执行过程如图表 2-6 所示：



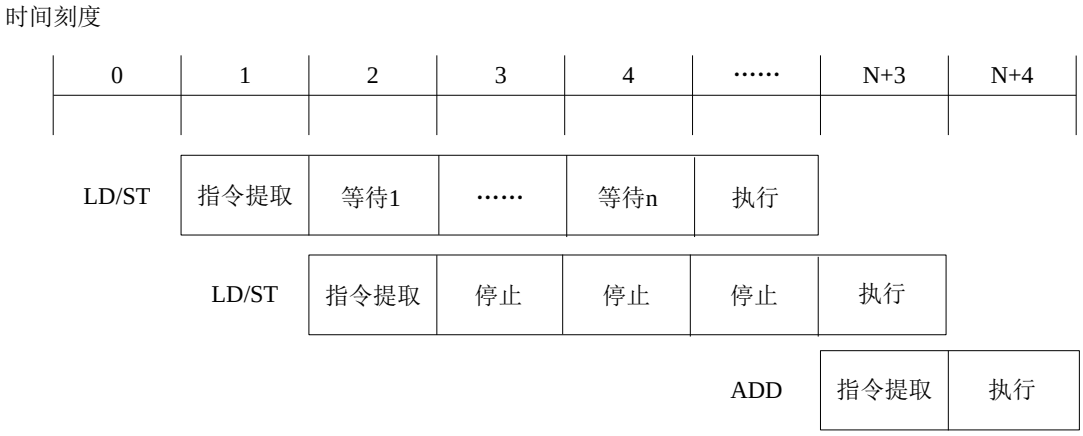
图表 2-6 JMP 指令执行过程

如果跳转指令的目标指令是字未对齐的 32 位指令时，取指需要两次访问指令总线，于是存在至少一个时钟周期的延迟。图表 2-7 显示了一条 BR 指令在目标指令是一条 32 位字未对齐指令 ADD 的执行过程：



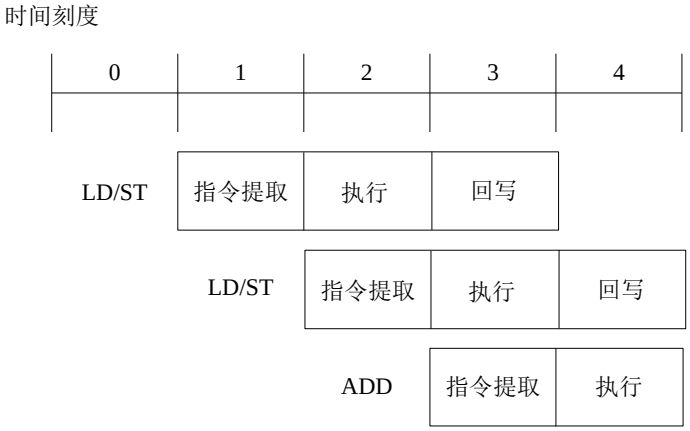
图表 2-7 跳转指令的目标指令是 32 位字未对齐指令的执行过程

对于访问存储区的指令，可能有等待状态。这会导致所有在访问存储区指令之后的指令处于停止状态，因为采用的是阻塞发射机制，必须等到访问存储区的指令完成之后这些指令才能执行。图表 2-8 显示了一条有等待状态的 ld/st 后跟一条无等待状态的 LD/ST 和一条单周期指令 ADD 的执行过程：



图表 2-8 带有等待状态的指令流水执行过程

对于配置 LOAD/STORE 有快速退休机制的 S802，且存储器访问都可以在单周期内完成，其时序图如图表 2-9 所示。



图表 2-9 具有快速退休功能的指令流水执行过程

2.3. 可信防护技术

S802 面向安全领域，设计了可信防护技术，可用于系统的安全防护，主要特征包括：

- 基于同一物理处理器核，虚拟化出两个世界，分别为安全世界和非安全世界；
- 支持软件形式对存储器和 I/O 空间进行两个世界的空间划分；
- 支持可信中断；
- 支持可信调试；
- 支持可信引导；

具体请参考《CK_TEE_Lite_用户手册》。

2.4. 紧耦合 IP 架构

为了提高 S802 的系统集成度，方便用户集成与开发，S802 实现了一系列与 CPU 核关系密切的系统 IP，这些 IP 统称为紧耦合 IP（Titly Coupled IP，TCIP）。S802 的紧耦合 IP 包括系统计时器 CoreTim、矢量中断控制器 VIC、功耗管理单元 PWRM、高速缓存控制寄

存器单元 CRU 和安全扩展单元 SEU。

矢量中断控制器的主要特征包括:

- 中断数量硬件可配置，支持 16/32 个中断源；
- 中断优先级软件可定义，可定义 4 个级别优先级；
- 支持硬件中断嵌套；
- 支持电平和脉冲两种中断源信号。

系统计时器的主要特征包括:

- 1 个 24 位的计数器；
- 支持输入时钟可选择，可以选择 CPU 时钟或外部输入时钟；
- 支持中断产生。

具体请参考《CSKY 处理器紧耦合 IP 用户手册》。

2.5. 抗物理攻击技术

为增强安全性，S802 处理器实现了一系列安全机制。这些安全机制耦合在处理器核、总线接口单元和片上存储器中，能够提升 S802 处理器在时间攻击、功耗分析攻击、错误注入和缓冲区溢出等主要攻击手段下的防护能力。在有效抵御攻击的同时，S802 处理器的安全机制硬件资源小，性能与功耗可控。

S802 处理器的安全扩展共包含 19 项安全机制。根据应用场景和资源需求，这些安全机制均可硬件独立配置。此外，对功耗与性能敏感的安全机制可以通过软件配置，在运行时开启、关闭和改变防护强度。S802 处理器安全机制和攻击防护类型下表所示。

安全机制	抗时间攻击	反功耗分析	抗错误注入	抗缓冲区溢出
1. 分支执行周期一致化	√			
2. 乘法执行周期一致化	√			
3. 随机执行周期	√	√		
4. 硬件随机指令		√		
5. 随机时钟噪声源		√		
6. 通用寄存器校验			√	
7. 控制寄存器校验			√	
8. 流水线校验			√	
9. 可配置校验算法			√	
10. 程序计数器校验			√	

11. 关键寄存器互补备份			√	
12. 数据通路极性翻转		√		
13. 保护区可执行检查				√
14. 显式内存访问属性				√
15. 独立堆栈指针				√
16. 外部通用寄存器复位			√	√
17. 安全违反检测与上报			√	√
18. 总线数据加扰		√		
19. 总线数据校验			√	

图表 2-10 安全机制和攻击防护类型

S802 处理器安全扩展单元与其它系统 IP 共享统一的内存地址空间，通过加载指令（Load）和存储指令（Store）进行寄存器访问和与功能控制。安全扩展单元的内存地址分配如下所示。

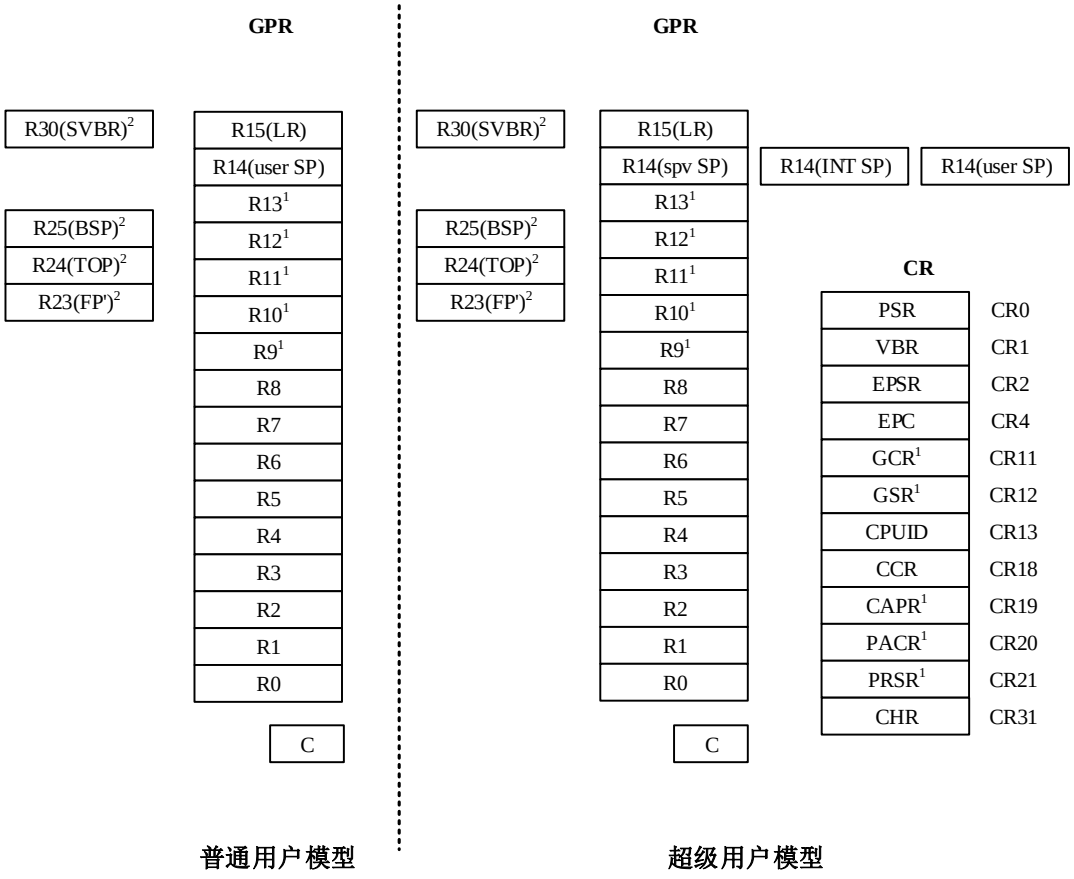
IP名	内存地址空间
安全扩展单元	0xE000EFA0-0xE000EFAF

图表 2-11 安全扩展单元的内存地址分配

S802 处理器的绝大部分安全机制依赖于随机属性，因此 S802 处理器安全扩展需要系统通过处理器接口信号输入 32 位真随机源。在调试模式下，所有安全机制被关闭，任何安全违反将被忽略。

3. 编程模型

3.1. 工作模式及寄存器视图



注：1、可配置资源；2、仅在二进制代码转译机制中实现

图表 3-1 编程模型

S802 定义了两种编程模式：超级用户模式和普通用户模式。当 PSR 内的 S 位被置位，处理器就在超级用户模式下执行程序。处理器复位后工作在超级用户模式下。

两种运行模式对应不同的操作权限，区别主要体现在两个方面：1）对控制寄存器的访问；2）特权指令的使用;3) 对紧耦合 IP 的控制寄存器访问。普通用户模式只允许访问通用寄存器；超级用户模式可以访问所有的通用寄存器和控制寄存器。用户程序因此可以避免接触特权信息，而操作系统通过协调与用户程序的行为来为用户程序提供管理和服务。超级用户模式下可以访问所有紧耦合 IP 的控制寄存器，用于调度 CPU 资源。

普通用户模式下可以访问普通用户堆栈指针（后续段落简称 user SP）。在普通用户模式下不能访问超级用户堆栈指针（后续段落简称 spv SP）和中断指针（INT SP）。

普通用户模式下可以访问链接寄存器（后续段落简称 LR），该 LR 与超级用户模式共享。

普通用户模式下，条件/进位位（C）位于 PSR 的最低位，可以被访问和更改，是 PSR 中唯一能在普通用户模式下被访问的数据位。

普通用户模式可以使用绝大多数的指令，除了对系统产生重大影响的特权指令如 STOP，DOZE，WAIT，MFCR，MTCR， RTE 之外。普通用户模式可以通过使用 TRAP #n

指令来进入超级用户模式。

超级用户模式下可以访问所有通用寄存器以及控制寄存器。在超级用户模式下可以访问 user SP 和 spv SP。如果用户需要在超级用户模式下访问 uesr SP，则需要使用 mtcr rx cr<14,1>和 mfcr rz cr<14,1>来实现，访问 int sp 则需要使用 mtcr rx cr<15,1>和 mfcr rz cr<15,1>来实现。

超级用户模式下可以使用 S802 支持的所有指令。

3.2. 通用寄存器

图表 3-2 列出了普通用户编程模式下的一些寄存器：

- 16 个 32 位通用寄存器（R15~R0）；
- 条件码 / 进位标志位（C bit）。
- 二进制转译堆栈栈底寄存器 R23（二进制代码转译机制时实现）
- 二进制转译堆栈栈顶寄存器 R24（二进制代码转译机制时实现）
- 二进制转译堆栈栈针寄存器 R25（二进制代码转译机制时实现）
- 软件向量基址寄存器 R30（二进制代码转译机制时实现）

名称	功能
R0	不确定，函数调用时第一个参数
R1	不确定，函数调用时第二个参数
R2	不确定，函数调用时第三个参数
R3	不确定，函数调用时第四个参数
R4	不确定
R5	不确定
R6	不确定
R7	不确定
R8	不确定
R9	不确定
R10	不确定
R11	不确定
R12	不确定
R13	不确定
R14(user)	堆栈指针（普通用户编程模式）
R15(user)	链接寄存器
R23(fp')	二进制转译堆栈栈底寄存器（二进制代码转译机制时实现）

R24(top)	二进制转译堆栈栈顶寄存器（二进制代码转译机制时实现）
R25(bsp)	二进制转译堆栈栈针寄存器（二进制代码转译机制时实现）
R30(svbr)	软件向量基址寄存器（二进制代码转译机制时实现）

C	条件码/进位标志

图表 3-2 普通用户编程模式寄存器

通用寄存器包含了指令操作数和结果以及地址信息。软硬件上约定这些通用寄存器做为子程序的链接调用，参数传递以及堆栈指针等功能。

S802 为普通用户模式和超级用户模式分别设计了堆栈指针 R14。普通用户模式只能访问用户程序的 R14（User SP）；在超级用户模式下，系统软件不仅可以访问系统程序的 R14（Spv SP），还可以访问普通用户程序的 R14（User SP）。超级用户模式下，对通用寄存器 R14 的索引将会使用超级用户模式的寄存器 R14(Spv SP)。若用户要在超级用户模式下访问 R14(User SP)，可通过 MFCR/MTCR 访问 CR<14,1>完成。S802 还为中断模式设置了专用的 R14（INT SP），该功能配有使能位，当使能时进入中断后会默认使用 INT SP。普通用户模式下无法访问 INT SP。超级用户模式下可以通过 CR<15,1>访问 INT SP。

3.2.1.条件码 / 进位标志位

条件码 / 进位标志位代表了一次操作后的进位或条件判断结果。条件码 / 进位标志位能够作为比较操作指令的结果被置位，或者作为另一些高精度算术或逻辑指令的结果被置位。另外，特殊的指令如 XTRB[0-3]等也会影响条件码 / 进位标志位的值。

3.2.2. 二进制代码转译模式

S802 实现了二进制代码转译模式(Binary Code Translation Mode)，加速 JAVA 等解释性语言的执行。二进制代码转译功能在普通 S802 核基础上增加了以下资源：

- 二进制转译堆栈栈底寄存器 FP'，位于第 23 号通用寄存器（R23）。
- 二进制转译堆栈栈顶寄存器 TOP，位于第 24 号通用寄存器（R24）。
- 二进制转译堆栈栈针寄存器 BSP，位于第 25 号通用寄存器（R25）。
- 软件矢量基址寄存器 SVBR，位于第 30 号通用寄存器（R30）。
- 二进制代码转译模式位 BM，位于 PSR[2]。
- 增加了 BMSET/BMCLR/JMPIX/BPUSH/BPOP 等指令。

普通用户设置 PSR 的 BM 位使得处理器运行于二进制代码转译模式，通过清除 BM 位使得处理器回到正常模式运行。相比正常模式，二进制代码转译模式将影响加载存储操作的运行。

在二进制代码转译模式下，处理器对加载存储操作的基地址进行合法性检查。一旦二进制堆栈访问溢出（仅限于 BPUSH/BPOP 指令），则内存访问操作不执行并抛出软件异常，处理器将下条指令的 PC 保存至链接寄存器 R15，同时从 SVBR-12（针对二进制堆栈访问溢出）地址获取跳转入口地址并跳转执行。如果二进制堆栈访问未溢出，则内存访问操作正常完成。通过硬件监测，避免了软件在二进制转译中通过软件比较的操作，提高运行速度。

二进制代码转译模式新增的指令具体见附件 B 的指令术语表。

3.3. 系统控制寄存器

系统程序员用超级用户编程模式来设置系统操作功能，I/O 控制，以及其他受限的操作。超级用户编程模式由通用寄存器和以下寄存器组成，如图表 3-3 所示：

- 1 个超级用户编程模式堆栈指针寄存器（R14）
- 1 个中断模式下堆栈指针寄存器（R14）*
- 处理器状态寄存器(PSR)；
- 向量基址寄存器(VBR)；
- 异常保留程序计数器(EPC)；
- 异常保留处理器状态寄存器(EPSR)；
- 32 位全控制寄存器(GCR)（可配置宽度）*；
- 32 位全状态寄存器(GSR)（可配置宽度）*；
- 产品序号寄存器(CPUIDR)；
- 内存保护配置寄存器(CCR)；
- 访问权限配置寄存器(CAPR) *；
- 保护区控制寄存器(PACR) *；
- 保护区选择寄存器(PRSR) *；
- 隐式操作寄存器（CHR）。

*注：可选择寄存器仅在特定配置时有效。

图表 3-3 超级用户编程模式附加资源

3.3.1.1. 处理器状态寄存器（PSR，CR<0,0>）

处理器状态寄存器（PSR）存储了当前处理器的状态和控制信息，包括 C 位，中断有

效位和其他控制位。在超级用户编程模式下，软件可以访问处理器状态寄存器（PSR）。处理器状态寄存器指示处理器处于超级用户模式或者普通用户模式（S 位）。同样也指出了异常保留寄存器是否可用来保存当前相应的内容，以及中断申请是否有效等。

	31	30						24	23							16
	S	0							VEC［7：0］							
Reset	1	0							0							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	00		0	0	0		MM	EE	IC	IE	0	0	0	*BM	0	C
Reset	0		0	0	0		0	0	0	0	0	0	0	0	0	0

图表 3-4 处理器状态寄存器

S-超级用户模式设置位：

当 S 为 0 时，处理器工作在普通用户模式；

当 S 为 1 时，处理器工作在超级用户模式；

该位在被 reset 和进入异常处理时由硬件置 1。

VEC[7:0]-异常事件向量值：

当异常出现时，这些位被用来计算异常服务程序向量入口地址，且会在被 reset 时清零。

MM-不对齐异常掩盖位：

当 MM 为 0 时，读取或存储的地址不对齐，处理器会响应该非对齐异常；

当 MM 为 1 时，读取或存储的地址不对齐，处理器不会响应该非对齐异常：若处理器硬件支持非对齐访问，则处理器使用该非对齐地址对存储器进行非对齐访问；若处理器硬件不支持非对齐访问，则处理器将该非对齐地址的低位强行置 0 后对存储器进行对齐访问。在任何情况下，只要多周期内存访问指令（如 STM、LDM、PUSH、POP、NIE、NIR、IPUSH、IPOP 等）发生地址非对齐，处理器都要响应非对齐异常。

未对齐的具体操作实现如下：

- 地址为 1，2, 3 的 word 读访问会在总线上出现两次 word 读操作，地址分别为 0 和 4。
- 地址为 1 的 half word 的读访问，会出现一次 word 读操作，地址为 0。
- 地址为 3 的 half word 的读访问，会出现两次 word 读操作，地址为 0 和 4。
- 地址为 1 的 word 写操作会在总线上出现地址为 1 的 byte 写，地址为 2 的 half word 写，地址为 4 的 byte 写。
- 地址为 2 的 word 写操作会在总线上出现地址为 2 的 half 写，地址为 4 的 half word 写。
- 地址为 3 的 word 写操作会在总线上出现地址为 3 的 byte 写，地址为 4 的 half word 写，地址为 6 的 byte 写。
- 地址为 1 的 half word 写操作会在总线上出现地址为 1 的 byte 写，地址为 2 的 byte 写。
- 地址为 3 的 half word 写操作会在总线上出现地址为 3 的 byte 写，地址为 4 的 byte 写。

该位会被 reset 清零。

EE-异常有效控制位：

当 EE 为 0 时，异常无效，此时除了普通中断之外的任何异常一旦发生，都会被 S802

认为是不可恢复的异常；

当 EE 为 1 时，异常有效，所有的异常都会正常的响应和使用 EPSR 与 EPC。

该位会被 reset 清零，也在处理器响应异常时被清零。

IC-中断控制位：

当 IC 为 0 时，中断只能在指令之间被响应；

当 IC 为 1 时，表明中断可在长时间、多周期的指令执行完之前被响应；

该位会被 reset 清零，不受其它异常影响。

IE-中断有效控制位：

当 IE 为 0 时，中断无效，EPC 和 EPSR 都无效；

当 IE 为 1 时，中断有效，（此时 EE 位也需要为 1，否则中断依然无效）；

该位会被 reset 清零，也在处理器响应异常时被清零。

***BM-二进制代码转译模式控制位：**

当 BM 为 0 时，处理器工作在正常模式；

当 BM 为 1 时，处理器工作在二进制代码转译模式，影响 LD/ST/BPUSH/BPOP 等指令的执行；

该位会被 reset 清零，不受其它异常影响。

注：该位在处理器配置了二进制代码转译机制时实现，若处理器不支持该机制，该位恒为 0。

C-条件码 / 进位位

该位用作条件判断位为一些指令服务。

该位会被 reset 清零。

3.3.1.2. 更新 PSR

PSR 可以通过几种不同的方式被更新，对 PSR 中控制位的更改所产生的影响也多种多样。PSR 通常可以通过异常响应，异常处理和执行 RTE，MTCR 指令被修改，这些修改的实现有四个方面。

- 异常响应和异常处理更新 PSR：

更新 PSR 是异常响应和异常服务程序入口地址计算中的一部分，它将更新 PSR 中 S，VEC，IE，EE 位。对 S，VEC，IE， EE 位的改动优先于异常服务程序向量入口地址的取址。对 VEC 位的改动优先于异常服务程序中的第一条指令的执行。

- RTE 指令更新 PSR：

更新 PSR 作为 rte 指令执行的一部分，可能会对 PSR 中的所有位都改动。其中对 S，IE，EE、BM 的改动优先于对返回 PC 的取址，对 VEC, MM, IC 和 C 位的改动优先于程序返回后第一条指令的执行。

- MTCR 指令更新 PSR：

若目标寄存器是 CR<0,0>的话，更新 PSR 将会作为 mtcr 指令执行的一部分。这种更新将可能会改变 PSR 中所有位的值，紧接着的指令、异常事件和中断响应将会采用新的 PSR 值。

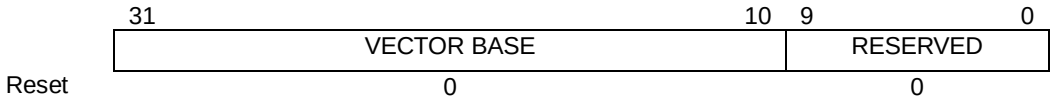
- BMCLR、BMSET 指令更新 PSR：

更新 PSR 作为 BMCLR 和 BMSET 指令执行的一部分，紧接着的指令、异常事件和中

断响应将会采用新的 PSR 值。

3.3.1.3. 向量基址寄存器（VBR，CR<1,0>）

VBR 寄存器用来保存异常向量的基址。该寄存器包含 22 个高位有效位，10 个保留位（其值为 0）。VBR 的复位值为 0X00000000。



图表 3-5 基址向量寄存器

3.3.1.4. 异常保留寄存器（CR<2,0>～CR<5,0>）

EPSR 和 EPC 这些寄存器在遇到异常情况时被用来保存当前处理器执行的内容。更详细的信息请参考第六章异常处理。

3.3.1.5. 全局控制寄存器（GCR，CR<11,0>）

全局控制寄存器是用来控制外部设备和事件。它通过芯片口上提供的平行输出接口实现指定控制。一般来说，可以通过简单设置 GCR 来管理功耗，设备控制，事件安排处理以及其它的基本的功能。至于 GCR 中每一位对应的控制功能，用户可以根据情况自行定义。全控制寄存器是可读可写的。在 S802 中全局控制寄存器的位宽是硬件可配置的。

3.3.1.6. 全局状态寄存器（GSR，CR<12,0>）

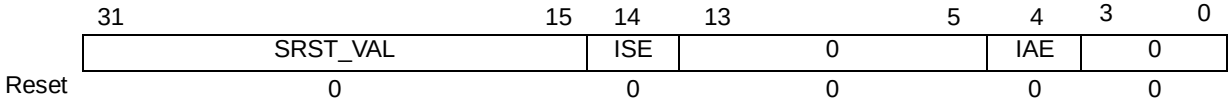
全局状态寄存器是用来标记外围设备和事件的。它通过芯片口上提供的输入接口将外部状态送入到 S802 内部，从而实现监测。一般来说，可以通过查看 GSR 来检测外围设备状态和事件。全状态寄存器是只读的。在 S802 中全局状态寄存器的位宽是硬件可配置的。

3.3.1.7. 产品序号寄存器（CPUIDRR，CR<13,0>）

该寄存器用于存放平头哥半导体有限公司产品的内部编号。产品序号寄存器是只读的，其复位值由产品本身决定。S802 产品序号寄存器版本 4.0 版，具体定义请参考《C-SKY 产品 ID 定义规范（4.0 版）》。

3.3.1.8. 隐式操作寄存器（CHR，CR<31,0>）

CR<31,0>用以实现处理器内各项隐式操作，在 S802 中，上述隐式操作包括：软件复位功能、中断响应加速功能。



图表 3-6 隐式操作寄存器

软件复位判定值 SRST_VAL：

当 SRST_VAL 域写入特定值，可实现处理器的软件复位，该特定值默认为 16'hABCD。

软件复位操作会使处理器向外发起一个系统时钟周期的复位请求信号，系统根据该复位请求信号复位处理器。

若处理器在正常运行模式下执行软件复位指令，则处理器将进入复位异常处理器程序（即异常向量号为零的异常服务程序）执行相应操作；若在调试模式下执行该软件复位指令，则处理器保持在调试模式，停在复位异常处理器程序（即异常向量号为零的异常服务程序）的第一条指令。

对该 SRST_VAL 域的读操作将无条件返回 0，另外对该寄存器写入除软件复位对应的特定值之外的任何其他值，将不产生任何效果。

上述软复位操作，需要在处理器异常使能位（即 PSR 中 EE 位）被置位时才能实现复位效果，否则该软复位操作将触发不可恢复异常。

中断指针使能位 ISE：

当 ISE 位为 1 时，中断指针被使能，在处理器发生任意中断（不含 tspending），即异常向量号大于等于 32 时，均使用该指针。

当 ISE 位为 0 时，中断指针没有被使能，在各个状态下均使用原指针。

当 Int_SP 没有被配置时，中断指针使能位默认为 0。

中断响应加速使能位 IAE：

当 IAE 位为 1 时，中断响应投机加速被使能，处理器将启动现场投机保存压栈，加速中断响应；

当 IAE 位为 0 时，中断响应加速不使能。

上述 IAE 位控制的中断响应加速机制，仅在硬件配置有中断嵌套加速指令 NIE、NIR、IPUSH、IPOP 时有效。

3.3.1.9. 其它控制寄存器

S802 的其它控制寄存器还包括：

- 内存保护配置寄存器(CCR)*；
- 访问权限配置寄存器(CAPR)*；
- 保护区控制寄存器(PACR)*；
- 保护区选择寄存器(PRSR)*；

其中，内存保护配置寄存器(CCR)、访问权限配置寄存器(CAPR)*、保护区控制寄存器(PACR)*、保护区选择寄存器(PRSR)*是和内存保护单元设置相关的控制寄存器，在 CPU 配置内存保护单元时有效，具体的控制寄存器定义请参考第六部分-内存保护。

3.3.2. 普通用户模式通用寄存器 14（R14(User SP)，CR<14,1>）

在超级用户模式下，普通用户模式通用寄存器 14 映射为控制寄存器 CR<14,1>，即超级用户通过访问 CR<14,1>可以访问普通用户模式堆栈指针寄存器。

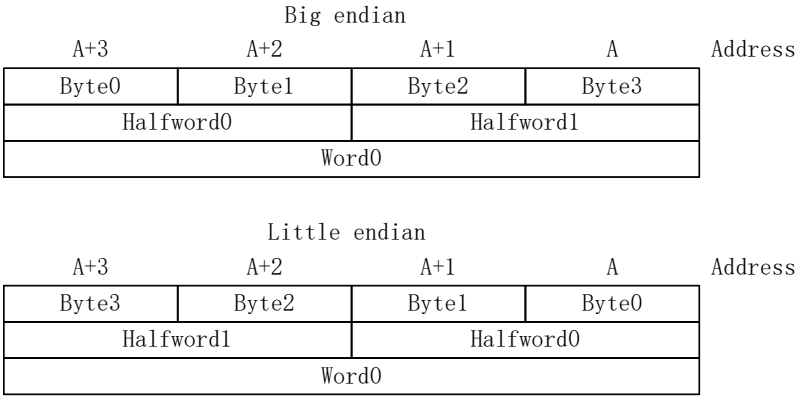
3.3.3. 中断指针寄存器（R14(Int_SP),CR<15,1>)

S802 可选择配置中断指针（Int_SP）用于在不同线程下共享中断堆栈空间，以节省堆栈总开销。如配置且 ISE 使能，则仅在中断（不含 tspending）中，即向量号大于等于 32 时使用该指针，硬件压栈及其余时刻均使用原指针。

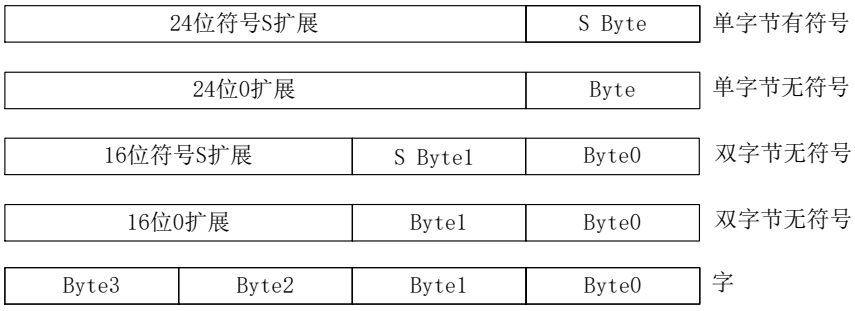
在非中断的超级用户态下，该寄存器映射为控制寄存器 CR<15,1>,即超级用户可通过

访问 CR<15,1>访问中断堆栈指针寄存器。

3.4. 数据大小端



图表 3-7 内存中的数据组织形式



图表 3-8 寄存器中的数据组织结构

S802 支持标准补码的 2 进制整数。每个指令操作数的长度可以明确地编码在程序中（load/store 指令），也可以隐含在指令操作中（index operation, byte extraction）。通常，指令使用 32 位操作数，产生 32 位结果。

S802 的存储器可以配置成大端模式或小端模式。在大端模式下，字 0 的最高位字节放在地址 0 上。而在小端模式（缺省模式）下，字 0 的最高位字节放在地址 3 上。在寄存器中，第 31 位是最高位。

3.5. 数据非对齐访问

S802 硬件可配置支持数据非对齐访问，具体参考 4.2.2 小节未对齐访问异常。

3.6. 系统地址映射

为了方便系统集成与开发，S802 对 4GB 的内存空间进行了地址划分与功能指定。总线矩阵单元对内存访问（指令或数据访问）地址进行仲裁并分发到不同的总线上。S802 推荐的系统地址划分及功能如图表 3-9 所示。为了让 SOC 设计更加灵活，芯片厂商也可以自己定义总线址空间的划分，通过配置 pad_bmu_iahbl_base ,pad_bmu_iahbl_mask 来达到目的。具体配置方法详见 8.1 节

图表 3-9 地址划分图

名称	内存地址空间	功能
指令总线	0x00000000-0x1FFFFFFF	存放指令
系统总线	0x20000000-0xDFFFFFFF	功能由系统开发者定义 可以存放指令、数据以及系统 IP
紧耦合 IP 总线	0xE0000000-0XEFFFFFFF	紧耦合 IP 的访问地址空间
系统总线	0xF0000000-0XFFFFFFFF	功能由系统开发者定义 可以存放指令、数据以及系统 IP

S802 的总线矩阵单元只根据内存访问地址进行仲裁，而不关心内存访问类型（指令访问/数据访问/紧耦合 IP 访问）。无论取指请求还是取数据请求都可以访问任一总线及存储器空间。编程者必须保证内存访问地址的正确性，以确保成功访问目标存储器。譬如：在配置了指令总线的 S802 中，如果指令访问地址落在了[0x20000000,0x40000000]，总线矩阵单元将把指令访问请求分发到系统总线上，并从系统总线存储器中访问。

S802 具有高度的可配置型，用户可以根据应用选择实现指令总线以及紧耦合 IP 总线的任意一个或多个，总线的配置方式对内存访问的影响如图表 3-10 所示

图表 3-10 地址划分图

名称	可配置性	配置情况对内存访问的影响	
指令总线	可配置	实现	位于[0x00000000-0x1FFFFFFF]的内存访问将分发到指令总线上
		不实现	位于[0x00000000-0x1FFFFFFF]的内存访问将分发到系统总线上
系统总线	不可配置	对系统总线空间的访问均被分发到系统总线上	
紧耦合IP总线	可配置	实现	位于[0xE0000000-0XEFFFFFFF]的内存访问将分发到紧耦合 IP 总线上
		不实现	位于[0xE0000000-0XEFFFFFFF]的内存访问将分发到系统总线上
系统总线	不可配置	对系统总线空间的访问均被分发到系统总线上	

以上对指令总线的访问只会发生在不可高缓的区域或者高速缓存缺失的情况下。

3.7. 内存访问顺序

S802 设计了多总线接口，系统的集成者可在总线上外接不同的存储器。由于不同总线上的存储器设备的访问延时不同，为了便于用户开发，S802 在硬件设计上保证了内存访问指令严格按照汇编指令的顺序依次完成，避免了用户以软件的方式保证内存访问的顺序。

譬如，如下两条指令序列，Ins A 访问系统总线的存储器，Ins B 访问指令总线的存储器。假设系统总线存储器的访问延时远大于指令总线存储器，为了保证指令按照程序的顺序完成，硬件保证 Ins A 指令执行完毕，才允许 Ins B 指令执行。

```
Ins A:          ld    r4, (r7)
```



```
                Ins B:          ld    r5, (r14)
```


4. 异常处理

异常处理(包括指令异常和外部中断)是处理器的一项重要技术，在某些异常事件产生时，用来使处理器转入对这些事件的处理。这些事件包括硬件错误、指令执行错误、和用户请求服务等等。本章主要描述异常种类、异常优先级、异常向量表、异常返回和总线错误恢复等内容。

4.1. 异常处理概述

异常处理是处理器根据内部或外部的异常事件从正常的程序处理转入特定的异常处理程序。引起异常的外部事件包括：外部设备的中断请求、读写访问错误和硬件重启；引起异常的内部事件包括：非法指令、非对齐错误（misaligned error）、特权异常，TRAP 和 BKPT 指令正常执行时也会产生异常。而且，非法指令、LD 和 ST 访问的地址没有对齐还有用户模式下执行特权指令都会产生异常。异常处理利用异常向量表跳转到异常服务程序的入口。

异常处理的关键就是在异常发生时，保存 CPU 当前指令运行的状态，在退出异常处理时恢复异常处理前的状态。异常能够在指令流水线的各个阶段被识别，并使后面的指令不会改变 CPU 的状态。异常在指令的边界上被处理，即 CPU 在指令退休时响应中断，并保存退出异常处理时下一条被执行的指令的地址。即使异常指令退休前被识别，异常也要在相应的指令退休时才会被处理。为了异常处理不影响 CPU 的性能，CPU 在异常处理结束后要避免重复执行以前的指令。S802 根据异常识别时的指令是否完成决定异常地址寄存器存储哪一条指令的地址。例如, 如果异常事件是外部中断服务请求，被中断的指令将正常退休并改变 CPU 的状态，它的下一条指令的地址（PC+2/PC+4，根据当前指令是 16 位或 32 位决定+2 或者+4）将被保存在异常地址寄存器（EPC）中作为中断返回时指令的入口；如果异常事件是由访问错误指令产生的，因为这条指令不能完成，它将异常退休但不改变 CPU 的状态(即不改变寄存器的值)，这条访问错误地址指令的地址（PC）将被保存在异常地址寄存器（EPC）中，CPU 从中断服务程序返回时继续执行这条访问错误指令。

异常按以下步骤被处理：

第一步，处理器保存 PSR 和 PC 到影子寄存器（EPSR 和 EPC）中。

第二步，将 PSR 中的超级用户模式设置位 S 位置 1（不管发生异常时处理器处于哪种运行模式），使处理器进入超级用户模式。

第三步，将 PSR 中的异常向量号 VEC 域更新为当前发生的异常向量号，标识异常类别以及支持共享异常服务的情况。

第四步，将 PSR 中的异常使能位 EE 位清零，禁止异常响应。在 EE 为零时发生的任何异常（除了普通中断），处理器都将其作为不可恢复错误异常处理。不可恢复的错误异常发生时，EPSR 和 EPC 也会被更新。

第五步，将 PSR 中的中断使能位 IE 位清零，禁止响应中断。

以上 2-4 步，同时发生。

第六步，处理器首先根据 PSR 中的异常向量号计算得到异常入口地址，然后用该地址获得异常服务程序的第一条指令的地址。将异常向量乘以 4 后加上异常向量基准地址（存在向量基准地址寄存器 VBR 中，当 VBR 不存在时该值恒为零）即得到异常入口地址，以该异常入口地址从存储器中读取一个字，并将该字的[31:1]装载到程序计数器中作为异常服务

程序的第一条指令的地址（PC 的最低位始终是 0，与异常向量表中取得的异常入口地址值的最低位无关）。对于向量中断，异常向量由外部的中断控制器提供；对于其它的异常，处理器根据内部逻辑决定异常向量。

最后一步，处理器从异常服务程序的第一条指令处开始执行并将 CPU 的控制权转交给异常服务程序，开始异常的处理。

所有的异常向量存放在超级用户地址空间，并以指令空间索引访问。在处理器地址映射中，只有重启向量是固定的。一旦处理器完成初始化，如果配置有 VBR，则允许异常向量表的基准地址被重载。

S802 支持 256 个字节的向量表包含 64 个异常向量（见图表 4-1）。开始的 30 号向量是用作在处理器内部识别的向量。31 号向量保留。其余的 32 个向量是留给外部设备的。外部设备通过 8 位的中断向量和中断请求使处理器响应中断服务。处理器响应中断请求时锁存这个中断向量。

图表 4-1 异常向量分配

向量号	向量偏移（十六进制）	向量分配
0	000	重启异常。
1	004	未对齐访问异常。
2	008	访问错误异常。
3	00C	保留。
4	010	非法指令异常。
5	014	特权违反异常。
6	018	保留。
7	01C	断点异常，地址观测异常
8	020	不可恢复错误异常。
9-15	024-03C	保留。
16－19	040－04C	陷阱指令异常（TRAP ＃0-3）。
20-21	050-054	保留。
22	058	Tspend 中断
23-30	05C-078	保留。
31	07C	保留
32－255	080－FC	保留给向量中断控制器使用。

4.2. 异常类型

本节描述外部中断异常和在 S802 内部产生的异常。S802 处理的异常有以下几类：

- 重启异常；

- 未对齐访问异常；
- 访问错误异常；
- 非法指令异常；
- 特权违反异常；
- 断点异常；
- 地址观测异常；
- 不可恢复错误异常；
- 陷阱指令异常；
- 软中断；
- 向量中断。

4.2.1.重启异常（向量偏移 0X0）

重启异常是所有异常中优先级最高的，它是用于系统初始化和发生重大故障后恢复系统。重启会中止处理器的所有操作，被中止的操作是不可恢复的。重启也在测试时用于初始化扫描链和时钟控制逻辑中锁存器的值，它也同时对处理器进行上电初始化。

重启异常设置 PSR（S）为高电平使处理器工作在超级用户模式。重启异常也会把 PSR（IE）和 PSR（EE）清零以禁止异常及中断响应。同时，VBR（向量基准寄存器）也被清零，异常向量表的基准地址就是 0X00000000，CPU 从异常向量表中以偏移地址 0X0 为偏移地址读取异常向量，并把它装载到程序计数器（PC）。异常处理器把控制权转移到 PC 指向的地址。

4.2.2.未对齐访问异常（向量偏移 0X4）

处理器试图在与访问大小不一致的地址边界上执行访问操作，就会发生地址未对齐访问异常。通过设置 PSR（MM），可以屏蔽该异常，屏蔽后处理器忽略对数据的对齐检查。MM 位被设置后，若处理器硬件支持非对齐访问，则处理器使用该非对齐地址对存储器进行非对齐访问；若处理器硬件不支持非对齐访问，则处理器将该非对齐地址的低位强行置 0 后对存储器进行对齐访问。EPC 指向试图进行未对齐访问的指令。S802 的未对齐访问异常只可能发生在数据访问上。

在任何情况下，如果拆分数据访问指令（如 LDM、STM、PUSH、POP、NIE、NIR、IPUSH、IPOP 等）发生地址非对齐，处理器都要响应非对齐访问异常。

4.2.3.访问错误异常（向量偏移 0X8）

如果外部总线接口访问错误返回信号（如 pad_biu_hresp[1:0]=1），就意味着访问发生了异常。当访问 MPU 保护的区域出现访问错误时，也会产生访问错误异常。当栈保护使能时，发生越栈访问，也会发生访问错误异常。EPC 指向该次总线请求对应的指令。

总线上的错误都会引起访问错误异常，使处理器进行异常处理。

4.2.4.非法指令异常（向量偏移 0X10）

如果在译码时发现了非法指令或没有实现的指令，S802 不会执行该指令，而是响应非法指令异常。EPC 指向该非法指令。

4.2.5.特权违反异常（向量偏移 0X14）

为了保护系统安全，一些指令被授予了特权，它们只能在超级用户模式下被执行。试图在用户模式下执行下面的特权指令都会产生特权违反异常：MFCR、MTCR、RTE、STOP、WAIT、DOZE。

处理器如果发现了特权违反异常，在执行该指令前进行异常处理。EPC 指向该特权指令。

4.2.6.断点异常（向量偏移 0X1C）

S802 在 HAD 控制和状态寄存器 CSR 的 FDB 位为 0 时执行到断点指令 BKPT 会响应断点异常。EPC 指向触发本次断点异常的 BKPT 指令。

4.2.7.地址观测异常（向量偏移 0X1C）

为了满足操作系统层面对应用的调试需求，并且脱离 jtag 的硬件限制，更自由的使用硬件调试功能，S802 在 HAD 控制和状态寄存器 CSR 的 MBEE 位为 1 时，内存硬断点的发生将产生异常请求，请求进入地址观测异常。

4.2.7.1. 响应过程

地址观测异常和其他异常响应大体类似，但也有一些区别，响应过程如下：

1. 将发生异常时的 PSR 更新到 EPSR 中。
2. 将 PC 保存到 EPC 中。需要注意的是，指令断点保存的是 current PC，而数据断点将保存 next PC。
3. 当 EE 位为 1 时，若满足地址观测异常响应条件，触发异常，异常向量号为 7
4. 当 EE 位为 0 时，若满足地址观测异常响应条件，不立即触发异常，而由硬件设置 DP 位表示有等待响应的地址观测异常，等待 EE 位被置 1 后延迟触发。
5. 异常优先级为最高。

4.2.7.2. 异常触发条件设置

该异常控制寄存器共用了 HAD 内部的硬件断点设置寄存器，该组寄存器可以通过 JTAG 接口进行读写，为了实现程序直接配置，CPU 内部将该组寄存器映射到了 TCIP 接口上。CPU 可以通过执行程序对相应地址进行读写来进行配置，无需再依靠 JTAG 端口。

具体配置地址观测异常相关内容请参考《紧耦合 IP 手册》。需要注意的是，TCIP 只有超级用户模式有访问权限。

4.2.8.不可恢复错误异常（向量偏移 0X20）

当 PSR（EE）为零时，除复位异常外的其他异常会产生不可恢复的异常，因为这时用于异常恢复的信息（存于 EPC 和 EPSR）可能由于不可恢复的错误而被重写。

由于软件在 PSR（EE）为零时应该排除了异常事件发生的可能，此不可恢复错误异常一般意味着有系统错误。在异常服务程序中，引起不可恢复错误异常的异常类型是不确定的。

4.2.9.陷阱指令异常（向量偏移 0X40－0X4C）

一些指令可以用来显示地产生陷阱异常。TRAP ＃n 指令可以强制产生异常，它可以用于用户程序的系统调用。在异常服务程序中，EPC 指向 TRAP 指令。

4.2.10. 软中断（向量偏移 0X58）

如果硬件配置有 VIC，CSKY CPU 支持软中断（TSpending 中断），异常向量号为 22，具体配置方法参考《CSKY E 系列处理器紧耦合 IP 手册》。

4.3. 中断异常

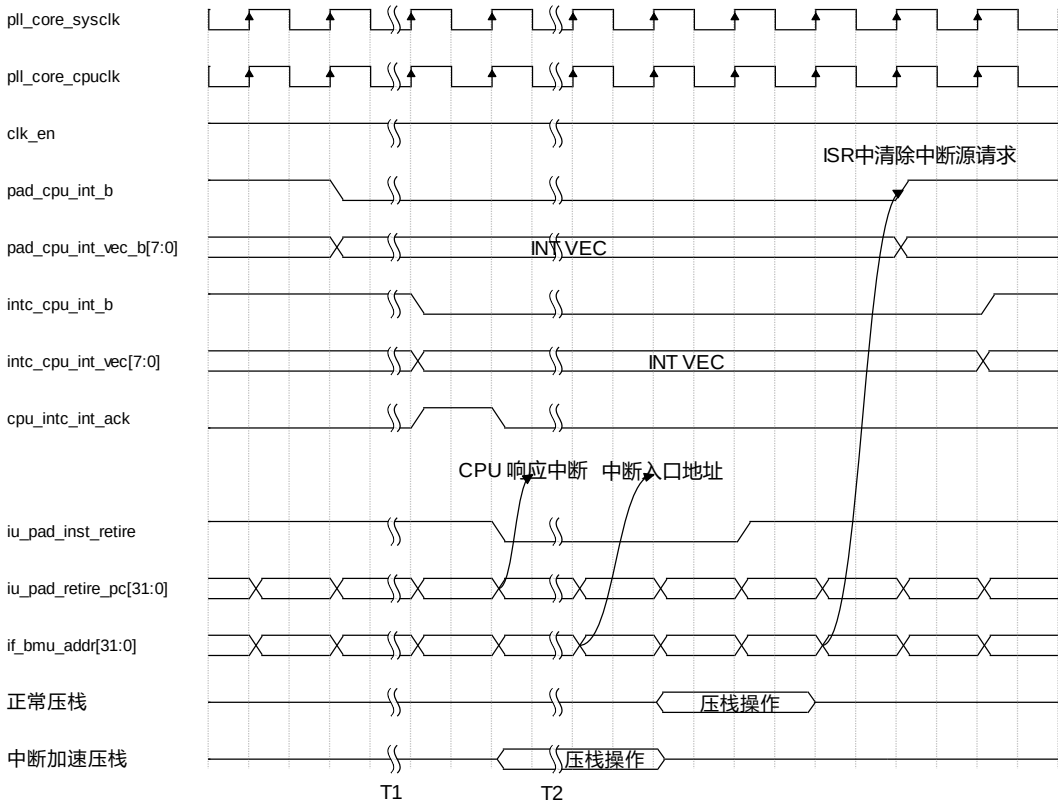
当外部设备需要向处理器请求服务或发送处理器需要的信息时，它可以用中断请求信号和相应的中断向量信号向处理器请求中断异常。

一般中断在指令的边界上被确认。如果 PSR（IC）位设置了，部分多周期指令包括 LDM、STM、PUSH、POP、IPUSH、IPOP、LDQ32、STQ32，可以被中断而不等它们完成，从而缩短中断响应延时。多周期指令 NIE 不可响应中断，NIR 只在指令执行的末尾响应中断，不能被 PSR（IC）位打断。

4.3.1.1. 向量中断（INT）

如果 PSR（IE）被清零，中断输入信号被屏蔽，处理器不响应异常。普通中断使用 EPSR 和 EPC 这一组异常影子寄存器，它也可以被 PSR（EE）屏蔽。当中断有效时，处理器通过专门的信号提供中断向量号，它可以是 32－255 中的任意一个（不允许使用 0－31）。

4.3.1.2. 中断处理过程



图表 4-2 中断处理过程

在上图中，当中断向量号已经准备好时，拉低 pad_cpu_int_b 中断信号线。如果没有配置 VIC 单元，则 sys_clk 采样一个周期(T1=1cyc)后拉低 intc_cpu_int_b 向 CPU 发送中断请求;如果配置有 VIC 则需要用 cpu_clk 经过两个周期进行采样和优先级的判断（T1=2cyc），之后才能向 CPU 发送中断请求。该信号经 CPU 内部时钟 cpu_clk 的上升沿采样后，CPU 内部收到中断，并根据向量信号取得中断向量。在外部系统均能在一个周期内响应的条件下，响应中断后，CPU 需要四个周期（T2=4cyc）才能发出中断服务程序第一条指令的取指令请求，进入中断服务程序。在中断服务程序中，应该由软件清除外部中断源，即拉高中断有效信号，此信号亦需 CPU 及外部两个时钟依次采样后才会退出中断。

S802 可配置中断加速功能。在 CPU 响应中断后，即开始投机执行 NIE,IPUSH 指令。如中断服务程序也首先执行这两条指令，则在外部系统均能在一个周期内响应的条件下能够节省五个周期的时间。如果发生投机预测错误，访问异常或调试请求，则中止中断加速功能。

如配置相应的中断控制器，则支持多个中断来源，可分别设置其对应的中断优先级并实现中断嵌套功能。更详细的中断机制及接口信号说明可参考集成手册以及紧耦合 IP 用户手册。

4.4. 异常优先级

如图表 4-3 所示，根据异常的特性和被处理的先后关系，S802 把优先级分为 5 级。在图表 4-3 中，1 代表最高优先级，5 代表了最低优先级。值得注意的是，在第 4,5 组中，几

个异常共享一个优先级，因为它们之间相互有排斥性。

在 S802 里，多个异常可以同时发生。重启异常是很特别的，它有最高的优先级。

所有其它的异常按图表 4-3 中的优先级关系进行处理。

如果 PSR（EE）被清零了，当异常发生时，处理器处理的是不可恢复异常。

如果多个异常同时发生，拥有最高优先级的异常最先被处理。处理器在异常返回后，重新执行产生异常的指令时，其余的异常可以重现。

图表 4-3 异常优先级

优先级	异常与它相关的优先级	特征
1	重启异常	处理器中止所有程序运行，初始化系统。
2	不对齐错误	在相关的指令退休后，处理器保存上下文并处理异常。
3	中断	如果 IC=0，中断在指令退休后被响应；如果 IC=1，处理器允许中断在指令完成之前就被响应。
4	不可恢复错误异常 访问错误	在相关的指令退休后，处理器保存上下文并处理异常。
5	非法指令 特权异常 陷阱指令 断点指令	在相关的指令退休后，处理器保存上下文并处理异常。

4.4.1.发生待处理的异常时调试请求

处理器如果在异常发生的同时接收到了调试请求信号，先进入调试模式。异常延后处理，直到处理器退出调试模式和产生异常的指令重新被执行。

4.5. 异常返回

根据正在处理的异常类型，处理器通过执行 rte 指令从异常服务程序中返回。rte 指令利用保存在 EPSR 和 EPC 影子寄存器中的上下文从异常服务程序中返回。

5. 指令集

5.1. 概述

S802 的指令集具有高级语言特性，并为一些频繁执行的指令进行了优化。该指令集包括标准的算术逻辑指令、位操作指令、字节提取指令、数据转移指令、控制流改变指令和条件执行指令，这些条件执行指令有助于减少短跳转的条件转移。S802 指令有两种宽度：16 位指令和 32 位指令，指令代码由两种指令混编而成，两种指令之间的切换没有额外的开销。

5.2. 32 位指令

本章主要介绍 S802 的 32 位指令集，包括 32 位指令集的功能分类、编码方式和寻址模式等。

5.2.1. 32 位指令功能分类

S802 的 32 位指令按照指令实现的功能来划分，可以分为：

- 数据运算指令
- 分支跳转指令
- 内存存取指令
- 特权指令
- 特殊功能指令

5.2.1.1. 数据运算指令

数据运算类指令可以进一步分为：

加减法指令：

图表 5-1 32 位加减法指令列表

ADDU32	无符号加法指令
ADDC32	无符号带进位加法指令
ADDI32	无符号立即数加法指令
SUBU32	无符号减法指令
SUBC32	无符号带借位减法指令
SUBI32	无符号立即数减法指令
RSUB32	反向减法指令
IXH32	索引半字指令
IXW32	索引字指令
INCF32	C 为 0 立即数加法指令
INCT32	C 为 1 立即数加法指令
DECF32	C 为 0 立即数减法指令
DECT32	C 为 1 立即数减法指令

逻辑操作指令：

图表 5-2 32 位逻辑操作指令列表

AND32	按位与指令
ANDI32	立即数按位与指令
ANDN32	按位非与指令
ANDNI32	立即数按位非与指令
OR32	按位或指令
ORI32	立即数按位或指令
XOR32	按位异或指令
XORI32	立即数按位异或指令
NOR32	按位或非指令
NOT32	按位非指令

移位指令：

图表 5-3 32 位移位指令列表

LSL32	逻辑左移指令
LSLI32	立即数逻辑左移指令
LSLC32	立即数逻辑左移至 C 位指令
LSR32	逻辑右移指令
LSRI32	立即数逻辑右移指令
LSRC32	立即数逻辑右移至 C 位指令
ASR32	算术右移指令
ASRI32	立即数算术右移指令
ASRC32	立即数算术右移至 C 位指令
ROTL32	循环左移指令
ROTLI32	立即数循环左移指令
XSR32	扩展右移指令

比较指令：

图表 5-4 32 位比较指令列表

CMPNEI32	立即数不等比较指令
CMPHSI32	立即数无符号大于等于比较指令
CMPLTI32	立即数有符号小于比较指令

数据传输指令：

图表 5-5 32 位数据传输指令列表

MOV32	数据传送指令

MOVF32	C 为 0 数据传送指令
MOVT32	C 为 1 数据传送指令
MOVI32	立即数数据传送指令
MOVIH32	立即数高位数据传送指令
MVC32	C 位传送指令
LRW32	存储器读入指令
GRS32	符号产生指令

比特操作指令：

图表 5-6 32 位比特操作指令列表

BCLRI32	立即数位清零指令
BSETI32	立即数位置位指令
BTSTI32	立即数位测试指令

提取插入指令：

图表 5-7 32 位提取插入指令列表

XTRB0.32	提取字节 0 并无符号展指令
XTRB1.32	提取字节 1 并无符号扩展指令
XTRB2.32	提取字节 2 并无符号扩展指令
XTRB3.32	提取字节 3 并无符号扩展指令

乘除法指令：

图表 5-8 32 位乘除法指令列表

MULT32	乘法指令

杂类运算指令：

图表 5-9 32 位杂类运算指令列表

FF0. 32	快速找 0 指令
FF1. 32	快速找 1 指令
BMASKI32	立即数位屏蔽产生指令
BGENI32	立即数位产生指令

5.2.1.2. 分支跳转指令

分支跳转指令可以进一步分为：

分支指令：

图表 5-10 32 位分支指令列表

BT32	C 为 1 分支指令
BF32	C 为 0 分支指令

跳转指令：

图表 5-11 32 位跳转指令列表

BR32	无条件跳转指令
BSR32	跳转到子程序指令
RTS32	链接寄存器跳转指令

5.2.1.3. 内存存取指令

内存存取指令可以进一步分为：

立即数偏移存取指令：

图表 5-12 32 位立即数偏移存取指令列表

LD32.B	无符号扩展字节加载指令
LD32.BS	有符号扩展字节加载指令
LD32.H	无符号扩展半字加载指令
LD32.HS	有符号扩展半字加载指令
LD32.W	字加载指令
ST32.B	字节存储指令
ST32.H	半字存储指令
ST32.W	字存储指令

多寄存器存取指令：

图表 5-13 32 位多寄存器存取指令列表

LDQ32	连续四字加载指令
LDM32	连续多字加载指令
STQ32	连续四字存储指令
STM32	连续多字存储指令

5.2.1.4. 特权指令

特权指令可以进一步分为：

控制寄存器操作指令：

图表 5-14 32 位控制寄存器操作指令列表

MFCR32	控制寄存器读传送指令
MTCR32	控制寄存器写传送指令

低功耗指令：

图表 5-15 32 位低功耗指令列表

WAIT32	进入低功耗等待模式指令
DOZE32	进入低功耗睡眠模式指令
STOP32	进入低功耗暂停模式指令

异常返回指令：

图表 5-16 32 位异常返回指令列表

RTE32	异常和普通中断返回指令

5.2.1.5. 特殊功能指令

特殊功能指令具体包括：

图表 5-17 32 位特殊功能指令列表

SYNC32	CPU 同步指令
TRAP32	无条件操作系统陷阱指令
BMSET32	BCTM 位置位指令
BMCLR32	BCTM 位清零指令

5.3. 16 位指令

本章主要介绍 S802 的 16 位指令集，包括 16 位指令集的功能分类，编码方式和寻址模式等。

5.3.1. 16 位指令功能分类

S802 的 16 位指令集按照指令实现的功能来划分，可以分为：

- 数据运算指令
- 分支跳转指令
- 内存存取指令

5.3.1.1. 数据运算指令

数据运算类指令可以进一步分为：

加减法指令：

ADDU16	无符号加法指令
ADDC16	无符号带进位加法指令
ADDI16	无符号立即数加法指令
SUBU16	无符号减法指令
SUBC16	无符号带借位减法指令
SUBI16	无符号立即数减法指令

图表 5-18 16 位加减法指令列表

逻辑操作指令：

AND16	按位与指令
ANDN16	按位非与指令
OR16	按位或指令
XOR16	按位异或指令
NOR16	按位或非指令
NOT16	按位非指令

图表 5-19 16 位逻辑操作指令列表

移位指令：

LSL16	逻辑左移指令
LSLI16	立即数逻辑左移指令
LSR16	逻辑右移指令
LSRI16	立即数逻辑右移指令
ASR16	算术右移指令
ASRI16	立即数算术右移指令
ROTL16	循环左移指令

图表 5-20 16 位移位指令列表

比较指令：

CMPNE16	不等比较指令
CMPNEI16	立即数不等比较指令
CMPHS16	无符号大于等于比较指令
CMPHSI16	立即数无符号大于等于比较指令
CMPLT16	有符号小于比较指令
CMPLTI16	立即数有符号小于比较指令
TST16	零测试指令
TSTNBZ16	无字节等于零寄存器测试指令

图表 5-21 16 位比较指令列表

数据传输指令：

MOV16	数据传送指令
MOVI16	立即数数据传送指令
MVCV16	C 位传送指令
LRW16	存储器读入指令

图表 5-22 16 位数据传输指令列表

比特操作指令：

BCLRI16	立即数位清零指令
BSETI16	立即数位置位指令
BTSTI16	立即数位测试指令

图表 5-23 16 位比特操作指令列表

提取插入指令：

ZEXTB16	字节提取并无符号扩展指令
ZEXTH16	半字提取并无符号扩展指令
SEXTB16	字节提取并有符号扩展指令
SEXTH16	半字提取并有符号扩展指令
REVB16	字节倒序指令
REVH16	半字内字节倒序指令

图表 5-24 16 位提取插入指令列表

乘除法指令：

MULT16	乘法指令

图表 5-25 16 位乘法指令列表

5.3.1.2. 分支跳转指令

分支跳转指令可以进一步分为：

分支指令：

BT16	C 为 1 分支指令
BF16	C 为 0 分支指令

图表 5-26 16 位分支指令列表

跳转指令：

BR16	无条件跳转指令
JMP16	寄存器跳转指令
JSR16	寄存器跳转到子程序指令
RTS16	链接寄存器跳转指令
JMPIX16	寄存器索引跳转指令

图表 5-27 16 位跳转指令列表

5.3.1.3. 内存存取指令

内存存取指令可以进一步分为：

立即数偏移存取指令：

LD16.B	无符号扩展字节加载指令
LD16.H	无符号扩展半字加载指令
LD16.W	字加载指令
ST16.B	字节存储指令
ST16.H	半字存储指令
ST16.W	字存储指令

图表 5-28 16 位立即数偏移存取指令列表

多寄存器存取指令：

POP16	出栈指令
IPOP16	中断出栈指令
PUSH16	压栈指令
IPUSH16	中断压栈指令
NIE16	中断嵌套使能指令
NIR16	中断嵌套返回指令

图表 5-29 16 位多寄存器存取指令列表

注：NIE16 和 NIR16 需要在特权模式下执行。

16 位二进制转译堆栈指令：

BPUSH16.H	二进制转译半字压栈指令
BPUSH16.W	二进制转译字压栈指令
BPOP16.H	二进制转译半字出栈指令
BPOP16.W	二进制转译字出栈指令

图表 5-30 16 位二进制转译堆栈指令

5.3.1.4. 特权指令

特权指令具体包括：

NIE16	中断嵌套使能指令
NIR16	中断嵌套返回指令

图表 5-31 16 位特权指令列表

注：NIE16 和 NIR16 同时也是多寄存器存取指令。

5.4. 指令集列表

S802 的指令集具有高级语言特性，并为一些频繁执行的指令进行了优化。该指令集包括标准的算术逻辑指令、位操作指令、字节提取指令、数据转移指令、控制流改变指令和条件执行指令，这些条件执行指令有助于减少短跳转的条件转移。

图表 5-32 列出了 S802 指令集中所有 16 位和 32 位指令 。

图表 5-32 S802 的指令集

汇编指令	32 位	16 位	汇编格式	指令描述
ADDU	○	○	ADDU16 RZ, RX ADDU32 RZ, RX, RY	无符号加法指令
ADDC	○	○	ADDC16 RZ, RX ADDC32 RZ, RX, RY	无符号带进位加法指令
ADDI	○	○	ADDI16 RZ, OIMM8 ADDI32 RZ, RX, OIMM12	无符号立即数加法指令
SUBU	○	○	SUBU16 RZ, RY SUBU32 RZ, RX, RY	无符号减法指令
SUBC	○	○	SUBC16 RZ, RY SUBC32 RZ, RX, RY	无符号带借位减法指令
SUBI	○	○	SUBI16 RZ, OIMM8 SUBI32 RZ, RX, OIMM12	无符号立即数减法指令
RSUB	○	×	RSUB32 RZ, RX, RY	反向减法指令
IXH	○	×	IXH32 RZ, RX, RY	索引半字指令
IXW	○	×	IXW32 RZ, RX, RY	索引字指令

INCF	○	×	INCF32 RZ, RX, IMM5	C 为 0 立即数加法指令
INCT	○	×	INCT32 RZ, RX, IMM5	C 为 1 立即数加法指令
DECF	○	×	DECF32 RZ, RX, IMM5	C 为 0 立即数减法指令
DECT	○	×	DECT32 RZ, RX, IMM5	C 为 1 立即数减法指令
AND	○	○	AND16 RZ, RX AND32 RZ, RX, RY	按位与指令
ANDI	○	×	ANDI32 RZ, RX, IMM12	立即数按位与指令
ANDN	○	○	ANDN16 RZ, RY ANDN32 RZ, RZ, RX	按位非与指令
ANDNI	○	×	ANDNI32 RZ, RX, IMM12	立即数按位非与指令
OR	○	○	OR16 RZ, RX OR32 RZ, RX, RY	按位或指令
ORI	○	×	ORI32 RZ, RX, IMM16	立即数按位或指令
XOR	○	○	XOR16 RZ, RX XOR32 RZ, RX, RY	按位异或指令
XORI	○	×	XORI32 RZ, RX, IMM12	立即数按位异或指令
NOR	○	○	XOR16 RZ, RX XOR32 RZ, RX, RY	按位或非指令
NOT	○	○	NOT16 RZ NOT32 RZ, RX	按位非指令
LSL	○	○	LSL16 RZ, RY LSL32 RZ, RX, RY	逻辑左移指令
LSLI	○	○	LSLI16 RZ, RX, IMM5 LSLI32 RZ, RX, IMM5	立即数逻辑左移指令
LSLC	○	×	LSLC32 RZ, RX, OIMM5	立即数逻辑左移至 C 位指令
LSR	○	○	LSR16 RZ, RY LSR32 RZ, RX, RY	逻辑右移指令
LSRI	○	○	LSRI16 RZ, RX, IMM5 LSRI32 RZ, RX, IMM5	立即数逻辑右移指令
LSRC	○	×	LSRC32 RZ, RX, OIMM5	立即数逻辑右移至 C 位指令
ASR	○	○	ASR16 RZ, RY ASR32 RZ, RX, RY	算术右移指令
ASRI	○	○	ASRI16 RZ, RX, IMM5 ASRI32 RZ, RX, IMM5	立即数算术右移指令
ASRC	○	×	ASRC32 RZ, RX, OIMM5	立即数算术右移至 C 位指令
ROTL	○	○	ROTL16 RZ, RY ROTL32 RZ, RX, RY	循环左移指令

ROTLI	○	×	ROTLI32 RZ, RX, IMM5	立即数循环左移指令
XSR	○	×	XSR32 RZ, RX, OIMM5	扩展右移指令
CMPNE	×	○	CMPNE16 RX, RY	不等比较指令
CMPNEI	○	○	CMPNEI16 RX, IMM5 CMPNEI32 RX, IMM16	立即数不等比较指令
CMPHS	×	○	CMPHS16 RX, RY	无符号大于等于比较指令
CMPHSI	○	○	CMPHSI16 RX, OIMM5 CMPHSI32 RX, OIMM16	立即数无符号大于等于比较指令
CMPLT	×	○	CMPLT16 RX, RY	有符号小于比较指令
CMPLTI	○	○	CMPLTI16 RX, OIMM5 CMPLTI32 RX, OIMM16	立即数有符号小于比较指令
TST	×	○	TST16 RX, RY	零测试指令
TSTNBZ	×	○	TSTNBZ16 RX	无字节等于零寄存器测试指令
MOV	○	○	MOV16 RZ, RX	数据传送指令
MOVF	○	×	MOVF32 RZ, RX	C 为 0 数据传送指令
MOVT	○	×	MOVT32 RZ, RX	C 为 1 数据传送指令
MOVI	○	○	MOVI16 RZ, IMM8 MOVI32 RZ, IMM16	立即数数据传送指令
MOVIH	○	×	MOVIH32 RZ, IMM16	立即数高位数据传送指令
LRW	○	○	LRW16 LABEL LRW16 IMM32 LRW32 LABEL LRW32 IMM32	存储器读入指令
MVCV	×	○	MVCV16 RZ	C 位取反传送指令
MVC	○	×	MVC32 RZ	C 位传送指令
BCLRI	○	○	BCLRI16 RZ, IMM5 BCLRI32 RZ, RX, IMM5	立即数位清零指令
BSETI	○	○	BSETI16 RZ, IMM5 BSETI32 RZ, RX, IMM5	立即数位置位指令
BTSTI	○	○	BTSTI32 RX, IMM5	立即数位测试指令
ZEXTB	×	○	ZEXTB16 RZ, RX;	字节提取并无符号扩展指令
ZEXTH	×	○	ZEXTH16 RZ, RX	半字提取并无符号扩展指令
SEXTB	×	○	SEXTB16 RZ, RX	字节提取并有符号扩展指令
SEXTH	×	○	SEXTH16 RZ, RX	半字提取并有符号扩展指令

XTRB0	○	×	XTRB0.32 RZ, RX	提取字节 0 并无符号扩展指令
XTRB1	○	×	XTRB1.32 RZ, RX	提取字节 1 并无符号扩展指令
XTRB2	○	×	XTRB2.32 RZ, RX	提取字节 2 并无符号扩展指令
XTRB3	○	×	XTRB3.32 RZ, RX	提取字节 3 并无符号扩展指令
REVB	×	○	REVB16 RZ, RX	字节倒序指令
REVH	×	○	REVH16 RZ, RX	半字内字节倒序指令
MULT	○	○	MULT16 RZ, RX MULT32 RZ, RX, RY	乘法指令
FF0	○	×	FF0.32 RZ, RX	快速找 0 指令
FF1	○	×	FF1.32 RZ, RX	快速找 1 指令
BMASKI	○	×	BMASKI32 RZ, OIMM5	立即数位屏蔽产生指令
BGENI	○	×	BGENI32 RZ, IMM5	立即数位产生指令
BT	○	○	BT16 LABEL BT32 LABEL	C 为 1 分支指令
BF	○	○	BF16 LABEL BF32 LABEL	C 为 0 分支指令
BR	○	○	BR16 LABEL BR32 LABEL	无条件跳转指令
BSR	○	×	BSR32 LABEL	跳转到子程序指令
JMP	×	○	JMP16 RX	寄存器跳转指令
JSR	×	○	JSR16 RX	寄存器跳转到子程序指令
GRS	○	×	GRS32 RZ, LABEL GRS32 RZ, IMM32	符号产生指令
RTS	x	○	RTS16	链接寄存器跳转指令
LD.B	○	○	LD16.B RZ, (RX, DISP) LD32.B RZ, (RX, DISP)	无符号扩展字节加载指令
LD.BS	○	×	LD32.BS RZ, (RX, DISP)	有符号扩展字节加载指令
LD.H	○	○	LD16.H RZ, (RX, DISP) LD32.H RZ, (RX, DISP)	无符号扩展半字加载指令
LD.HS	○	×	LD32.HS RZ, (RX, DISP)	有符号扩展半字加载指令
LD.W	○	○	LD16.W RZ, (RX, DISP) LD32.W RZ, (RX, DISP)	字加载指令
ST.B	○	○	ST16.W RZ, (RX, DISP) ST32.W RZ, (RX, DISP)	字节存储指令

ST.H	○	○	ST16.H RZ, (RX, DISP) ST32.H RZ, (RX, DISP)	半字存储指令
ST.W	○	○	ST16.W RZ, (RX, DISP) ST32.W RZ, (RX, DISP)	字存储指令
LDM	○	×	LDM32 RY-RZ, (RX)	连续多字加载指令
STM	○	×	STM32 RY-RZ, (RX)	连续多字存储指令
PUSH	×	○	PUSH16 REGLIST	压栈指令
POP	×	○	POP16 REGLIST	出栈指令
*BPUSH.H	×	○	BPUSH.H RZ	二进制转译半字压栈指令
*BPUSH.W	×	○	BPUSH.W RZ	二进制转译字压栈指令
*BPOP.H	×	○	BPOP.H RZ	二进制转译半字出栈指令
*BPOP.W	×	○	BPOP.W RZ	二进制转译字出栈指令
**IPUSH	×	○	IPUSH16	中断压栈指令
**IPOP	×	○	IPOP16	中断出栈指令
MFCR	○	×	MFCR32 RZ, CR<X, SEL>	控制寄存器读传送指令
MTCR	○	×	MTCR32 RX, CR<Z, SEL>	控制寄存器写传送指令
WAIT	○	×	WAIT32	进入低功耗等待模式指令
DOZE	○	×	DOZE32	进入低功耗睡眠模式指令
STOP	○	×	STOP32	进入低功耗暂停模式指令
RTE	○	×	RTE32	异常和普通中断返回指令
SYNC	○	×	SYNC32	CPU 同步指令
BKPT	×	○	BKPT16	断点指令
TRAP	○	×	TRAP32 0 TRAP32 1 TRAP32 2 TRAP32 3	无条件操作系统陷阱指令
**NIE	×	○	NIE16	中断嵌套使能指令
**NIR	×	○	NIR16	中断嵌套返回指令
*BMSET	○	×	BMSET32	BM 位置位指令
*BMCLR	○	×	BMCLR32	BM 位清零指令

*JMPIX	×	○	JMPIX16 RX, IMM	寄存器索引跳转指令

注：○表示相应指令集中存在该指令，×表示相应指令集中不存在该指令。*表示二进制代码转译机制增加的指令，**表示中断嵌套增强指令。

5.5. 指令执行延迟

图表 5-33 指令执行延时表

指令类型	指令	执行周期	备注
加减法指令	ADDU32/16	1	
	ADDC32/16	1	
	ADDI32/16	1	
	SUBU32/16	1	
	SUBC32/16	1	
	SUBI32/16	1	
	RSUB32	1	
	IXH32	1	
	IXW32	1	
	INCF32	1	
	INCT32	1	
	DECF32	1	
	DECT32	1	
逻辑操作指令	AND32/16	1	
	ANDN32/16	1	
	ANDI32	1	
	ANDNI32	1	
	OR32/16	1	
	ORI32	1	
	XOR32/16	1	
	XORI32	1	
	NOR32/16	1	
	NOT32/16	1	
移位指令	LSL32/16	1	
	LSLI32/16	1	
	LSLC32	1	

	LSR32/16	1	
	LSRI32/16	1	
	LSRC32	1	
	ASR32/16	1	
	ASRI32/16	1	
	ASRC32	1	
	ROTL32/16	1	
	ROTLI32	1	
	XSR32	1	
比较指令	CMPNE16	1	
	CMPNEI32/16	1	
	CMPHS16	1	
	CMPHSI32/16	1	
	CMPLT16	1	
	CMPLTI32/16	1	
	TST16	1	
	TSTNBZ16	1	
数据传输指令	MOV32/16	1	
	MOVF32	1	
	MOVT32	1	
	MOVI32/16	1	
	MOVIH32	1	
	MVCV32/16	1	
	MVC32	1	
	LRW32/16*	1	
	GRS32	1	
比特操作指令	BCLRI32/16	1	
	BSETI32/16	1	
	BTSTI32/16	1	
提取插入指令	ZEXTB16	1	
	ZEXTH16	1	
	SEXTB16	1	
	SEXTH16	1	

	XTRB0.32	1	
	XTRB1.32	1	
	XTRB2.32	1	
	XTRB3.32	1	
	REVB16	1	
	REVH16	1	
乘除法指令	MULT32/16	3-34	配置为快速乘法器时固定为 1 个周期
杂类运算指令	FF0. 32	1	
	FF1. 32	1	
	BMASKI32	1	
	BGENI32	1	
分支指令	BT32/16	1	
	BF32/16	1	
跳转指令	BR32/16	1	
	BSR32	1	
	JMPIX16	1	
	JMP16	1	
	JSR16	1	
	RTS32/16	1	
立即数偏移存取指令	LD32/16.B*	1	
	LD32.BS*	1	
	LD32/16.H*	1	
	LD32.HS*	1	
	LD32/16.W*	1	
	ST32/16.B*	1	
	ST32/16.H*	1	
	ST32/16.W*	1	
多寄存器存取指令	LDQ32*	4	拆分为 4 条 LD.W 指令。
	LDM32*	N	拆分为 N 条 LD.W 指令。
	STQ32*	4	拆分为 4 条 ST.W 指令。
	STM32*	N	拆分为 N 条 ST.W 指令。
	PUSH16*	1+N	拆分为 SUB 和 N 条 ST.W 指令。

	POP16*	2+N	拆分为 N 条 LD.W、ADD 和 RTS 指令。
	IPOP16*	7	拆分为 7 条原子指令
	IPUSH16*	7	拆分为 7 条原子指令
	NIE16*	5	拆分为 5 条原子指令
	NIR16*	5	拆分为 5 条原子指令
二进制转译堆栈指令	BPUSH16.H*	1	
	BPUSH16.W*	1	
	BPOP16.H*	1	
	BPOP16.W*	1	
控制寄存器操作指令	MFCR32	1	
	MTCR32	1	
低功耗指令	WAIT32	1	
	DOZE32	1	
	STOP32	1	
异常返回指令	RTE32	1	
特殊功能指令	SYNC32	1	
	BMSET32	1	
	BMCLR32	1	
	TRAP32	1	
	BKPT16	1	

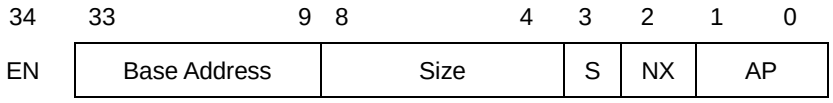
注： *表示内存存取相关指令，指令完成周期取决于总线延时或者高速缓存命中情况，表中数值所列为最快情况。

6. 内存保护

6.1. 内存保护单元简介

在受保护的系统中，主要有两类资源的访问需要被监视：存储器系统和外围设备。内存保护单元负责对存储器系统（包括外围设备）的访问合法性进行检查，其主要功能包括：1）判定当前工作模式下 CPU 是否具备对内存地址的读/写访问权限。2）获取该访问地址的附加属性，包括安全属性，是否可执行等。

内存保护单元支持 N 个表项（N 为 1-8 硬件可配置），可对 N 个区域的访问权限和属性进行设置。每个表项通过 0-7 的号码来标识和索引。内存保护单元的表项内容如下：



图表 6-1 内存保护单元表项

其中：

EN：表示该区域是否生效；

Base Address：表示该区域的起始地址；

Size：表示该区域的大小；

S：表示该区域的安全属性；

NX：表示该区域取指的可执行性；

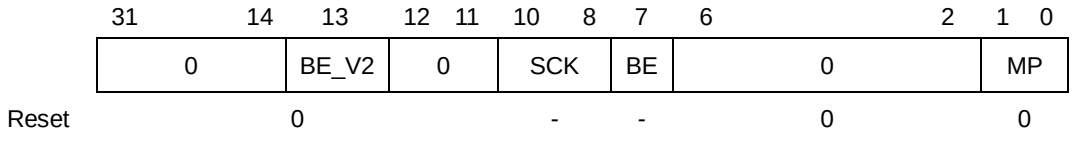
AP：表示该区域的访问权限；

具体的每个字段的属性参考 6.2 的系统控制寄存器部分。

6.2. 相关系统控制寄存器

6.2.1. 内存保护配置寄存器（CCR，CR<18,0>）

内存保护配置寄存器用来配置内存保护区，Endian 模式，以及系统和处理器的时钟比。



图表 6-2 内存保护配置寄存器

BE_V2-V2 版本大小端：

当 BE_V2 为 0 时，非 V2 版本大小端；

当 BE_V2 为 1 时，V2 版本大小端；

该位与 BE 一起决定处理器具体工作的大小端模式，该位仅在 BE 为 1 时起作用；

BE_V2 在 power on reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

SCK-系统和处理器的时钟比：

该位用来表示系统和处理器的时钟比，其计算公式为：时钟比＝SCK＋1，CPU 上有对应引脚引出。SCK 在 power on reset 时被配置且不能在之后改变。

该域目前没有任何功能，只供软件查询。

BE-Endian 模式：

当 BE 为 0 时，小端；

当 BE 为 1 时，大端；

BE 在 power on reset 时被配置且不能在之后改变，CPU 上有对应引脚引出。

MP-内存保护设置位：

MP 用来设置 MPU 是否有效，如下表：

MP	功能
00	MPU 无效
01	MPU 有效

图表 6-3 S802 内存保护设置

6.2.2.访问权限配置寄存器（CAPR，CR<19,0>）

CAPR 的各位如下图所示：

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	S7	S6	S5	S4	S3	S2	S1	S0	AP7		AP6		AP5		AP4	
Reset	0	0	0	0	0	0	0	0	0		0		0		0	

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	AP3		AP2		AP1		AP0		NX7	NX6	NX5	NX4	NX3	NX2	NX1	NX0
Reset	0		0		0		0		0	0	0	0	0	0	0	0

图表 6-4 访问权限配置寄存器

NX0~NX7-不可执行属性设置位：

当 NX 为 0 时，该区为可执行区；

当 NX 为 1 时，该区为不可执行区。

注：当处理器取指访问到不可执行的区域时，会出现访问错误异常。

S0~S7-安全属性设置位：

- ✓ 当 S 为 0 时，该区为非安全区；
- ✓ 当 S 为 1 时，该区为安全区。

AP0~AP7-访问权限设置位：

AP	超级用户权限	普通用户权限

00	不可访问	不可访问
01	读写	不可访问
10	读写	只读
11	读写	读写

图表 6-5 访问权限设置

6.2.3.保护区控制寄存器（PACR，CR<20,0>）

PACR 的各位如下图所示：

31 7	6	5 1	0
Base Address	0	Size	E
Reset -	0	-	0

图表 6-6 保护区控制寄存器

Base Address-保护区地址的高位：

该寄存器指出了保护区地址的高位，但写入的基地址必须与设置的页面大小对齐，例如设置页面大小为 8M，CR<20,0>[22:7]必须为 0，各页面的具体要求见下表：图表 6-7 保护区大小配置和其对基址要求。

Size-保护区大小：

保护区大小从 128B 到 4GB，它可以通过公式：保护区大小＝2^（Size+1）计算得到。因此 Size 可设置的范围为 00110 到 11111，其它一些值都会造成不可预测的结果。

Size	保护区大小	对基地址低位的要求
00000—00101	保留	—
00110	128B	无要求
00111	256B	CR<20,0>.bit[7]=0
01000	512B	CR<20,0>.bit[8:7]=0
01001	1KB	CR<20,0>.bit[9:7]=0
01010	2KB	CR<20,0>.bit[10:7]=0
01011	4KB	CR<20,0>.bit[11:7]=0
01100	8KB	CR<20,0>.bit[12:7]=0
01101	16KB	CR<20,0>.bit[13:7] =0
01110	32KB	CR<20,0>.bit[14:7] =0
01111	64KB	CR<20,0>.bit[15:7] =0
10000	128KB	CR<20,0>.bit[16:7] =0

Size	保护区大小	对基地址低位的要求
10001	256KB	CR<20,0>.bit[17:7] =0
10010	512KB	CR<20,0>.bit[18:7] =0
10011	1MB	CR<20,0>.bit[19:7] =0
10100	2MB	CR<20,0>.bit[20:7] =0
10101	4MB	CR<20,0>.bit[21:7] =0
10110	8MB	CR<20,0>.bit[22:7] =0
10111	16MB	CR<20,0>.bit[23:7] =0
11000	32MB	CR<20,0>.bit[24:7] =0
11001	64MB	CR<20,0>.bit[25:7] =0
11010	128MB	CR<20,0>.bit[26:7] =0
11011	256MB	CR<20,0>.bit[27:7] =0
11100	512MB	CR<20,0>.bit[28:7] =0
11101	1GB	CR<20,0>.bit[29:7] =0
11110	2GB	CR<20,0>.bit[30:7] =0
11111	4GB	CR<20,0>.bit[31:7] =0

图表 6-7 保护区大小配置和其对基址要求

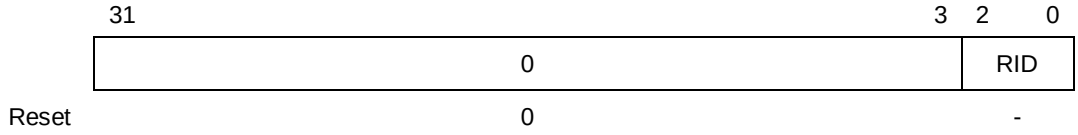
E-保护区有效设置：

当 E 为 0 时，保护区无效；

当 E 为 1 时，保护区有效。

6.2.4. 保护区选择寄存器（PRSR，CR<21,0>）

PRSR 用来选择当前操作的保护区，其各位如下图所示：



图表 6-8 保护区选择寄存器

RID-保护区索引值：

RID 表示所选择的对应的保护区，如 000 表示第 0 保护区。

6.3. 内存访问处理

当 MPU 被使能后，在内存访问信号产生时，MPU 会检查当前访问的地址是否在这些

保护区内：

如果访问的地址不在这些区中的任何一个，此内存访问会中途停止；

如果访问的地址在这些区中的一个或多个内，此访问被已使能的最高索引区（7 为最高，0 为最低）所控制。

6.4. 内存保护单元设置

6.4.1. 内存保护单元使能

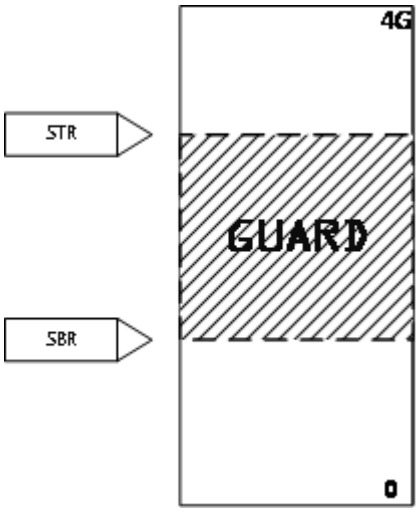
CR<20,0>中的第 0 位是 MPU 有效控制位。在 MPU 有效之前，至少有一个区被指定以及它相应的 NX、S 和 AP 也必须被设置。此外，这个让 MPU 有效的指令必须在指令地址访问有效的范围之内，即此指令所在的区域不可以在 MPU 中设置为拒绝访问。若不这么做，将会导致不可预测的结果。

6.4.2. 内存访问起始地址设置

CR<20,0>中定义了四/八个保护区的起始地址和大小。保护区大小必须是 2 的幂，能从 128B 到 4GB。起始地址必须与区大小对齐，比如：8KB 大小的保护区，起始地址可以是 32'h12346000，但是 16KB 大小的保护区，起始地址就不可以为这个值，可以是 32'h12344000。

6.5. 堆栈保护

S802 实现了可配置的堆栈保护功能。由栈顶寄存器（STR）、栈底寄存器（SBR）划出栈保护区间。该功能使能时会对所有以堆栈指针（R14）为基地址的存储操作进行检查，当访问地址超出这一区间时，发生访问错误异常。该异常优先级及处理方法与其他访问错误异常相同。



图表 6-9 堆栈保护示意图

堆栈保护机制的控制信息可通过 SGCR 寄存器进行设置。栈顶和栈底信息可通过 SGTR（栈顶）和 SGBR（栈底）进行设置；在配置有中断指针且使能时，只要打开堆栈保护功能，中断堆栈的栈保护功能也会被打开，在使用中断指针时进行检查。中断栈保护的上下边界可通过 SGISTR（中断栈保护上边界）和 SGISBR（中断栈保护下边界）进行设置。

6.5.1. 堆栈保护的相关寄存器

6.5.1.1. 栈保护控制寄存器（SGCR，CR<0,4>）

栈保护控制寄存器用来控制栈保护机制的使能，以及在何种用户模式下有效。

	31 … 2	1	0
	Preserved	SGP	SGE
Reset	0	0	0

图表 6-10 栈保护控制寄存器

SGE – 栈保护功能使能位

栈保护功能有效开关。在开启功能前，应将 SGTR 与 SGBR 配置完毕。功能开启后，匹配 CPU 所处状态后，CPU 将对以 SP（R14）作为基地址的内存访问指令的访问地址检查。当 SP< SGBR,或者 SP+size(byte:1,hw:2,word:4，以此类推)> SGTR，将发生访问错误异常（向量号：2）。此异常受到 EE 位控制，在 EE 未开启的情况下进入不可恢复异常。**SGE** 位默认值为 0。指令包括（LD/ST/POP/PUSH/STR/LDR 等）

0：栈保护功能关闭。

1：栈保护功能使能。

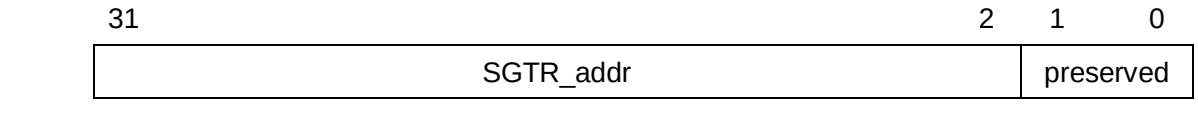
SGP – 超级用户、普通用户模式有效位

指示在超级用户、普通用户模式下，栈保护功能有效。**SGP** 位默认值为 0。

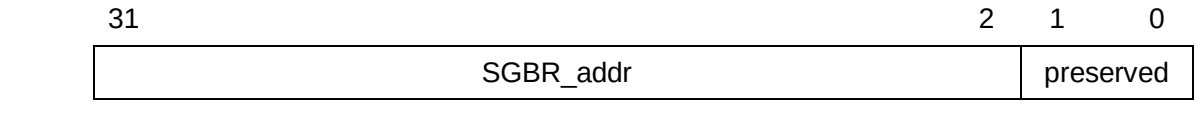
0：普通用户模式下有效。

1：超级用户模式下有效。

6.5.1.2. 栈保护上下边界寄存器（SGTR,CR<1,4>; SGBR,CR<2,4>）



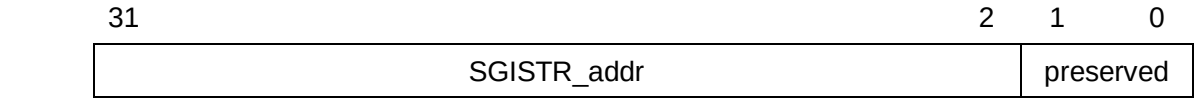
图表 6-11 栈保护上边界寄存器



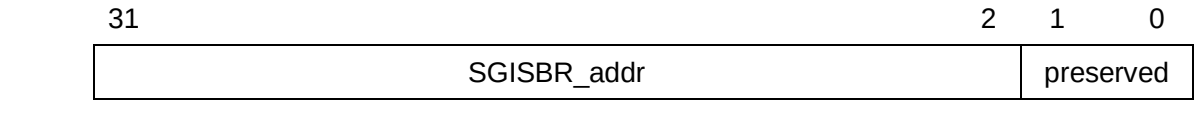
图表 6-12 栈保护下边界寄存器

SGTR_addr,SGBR_addr 分别表示栈保护上下边界。

6.5.1.3. 中断栈保护上下边界寄存器（SGISTR,CR<6,4>; SGISBR,CR<7,4>）



图表 6-13 中断栈保护上边界寄存器



图表 6-14 中断栈保护下边界寄存器

SGISTR_addr,SGISBR_addr 分别表示中断栈保护的上下边界。要使能中断堆栈保护需要先使能中断栈。

7. 安全机制详细介绍

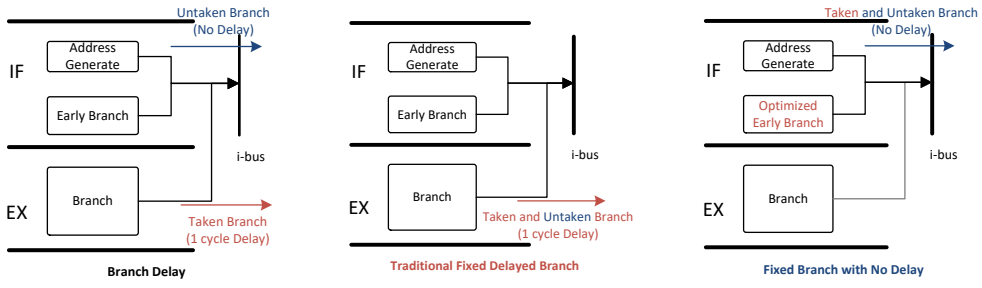
本章分别介绍各项安全机制的简要原理、参考指标、配置及使用说明。

7.1. 分支执行周期一致化

抗攻击类型：	
时间攻击	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[11]：FB	使能位：分支执行周期一致化使能
相关信号：	
无	

图表 7-1 安全机制信息

时间攻击通过观察分支指令的执行周期，推断分支指令是否被选中，进而获取指令流信息。传统安全处理器采用延迟获取未选中分支目标指令的方法，隐藏分支选中信息。S802 采用优化的分支预处理逻辑，实现无延时的分支执行逻辑。该预处理逻辑可以确保绝大多数分支指令在最快周期内发起分支目标指令的取值请求，与分支选中无关。分支执行周期一致化机制不依赖于预测，不仅能够固定分支执行周期，而且可以确保分支指令执行处于最优性能。



图表 7-2 分支执行周期一致化示意图

在特定指令序列和总线时序下，S802 处理器的部分分支指令仍然会因为选中与否呈现出不同的分支时序。因此分支执行周期一致化机制可以被使能，确保在上述情况下分支执行

周期的一致性。

该安全机制通过安全扩展单元控制寄存器的 BP 位（SECR[11]）控制。BP 位置 1 将开启分支执行周期一致化。BP 位置 0 将关闭分支执行周期一致化。

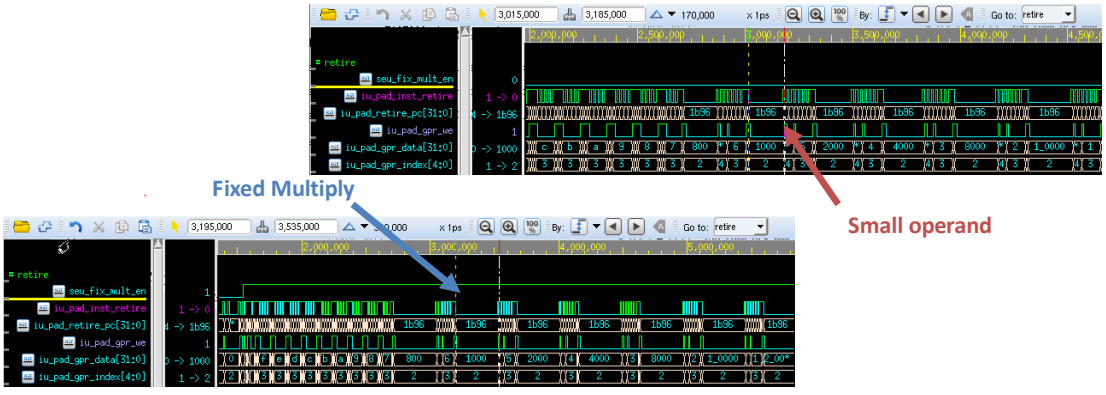
调试模式下分支执行周期一致化被关闭，且无法被使能。

7.2. 乘法执行周期一致化

抗攻击类型：	
时间攻击	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元，需配置最小化乘法器）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[6]：FM	使能位：乘法执行周期一致化使能
相关信号：	
无	

图表 7-3 安全机制信息

当 S802 处理器配置最小化乘法器时，默认根据操作数加速乘法执行。时间攻击通过观察乘法执行周期，可以获取操作数信息。乘法执行周期一致化机制将乘法执行周期与操作数解耦合，固定乘法执行的迭代次数。S802 处理器的乘法在其它乘法器配置下，执行周期固定，无需乘法执行周期一致化机制。



图表 7-4 乘法执行周期一致化示意图

该安全机制通过安全扩展单元控制寄存器的 FM 位（SECR[6]）控制。FM 位置 1 将开启乘法执行周期一致化。FM 位置 0 将关闭乘法执行周期一致化。

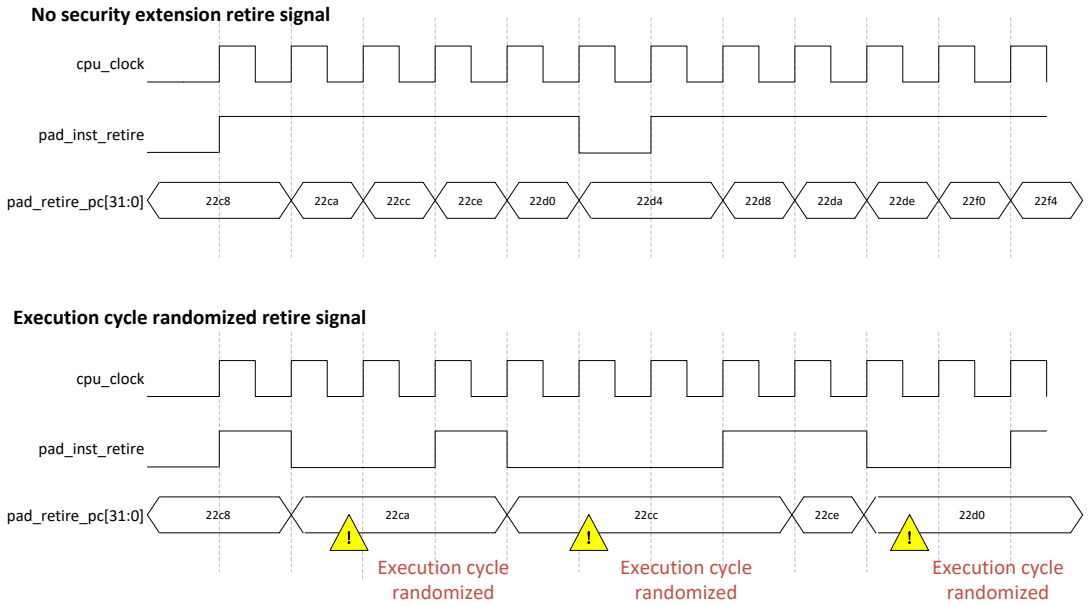
调试模式下乘法执行周期一致化被关闭，且无法被使能。

7.3. 随机执行周期

抗攻击类型：	
时间攻击、差分功耗分析	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[8]：RBI	使能位：随机执行周期使能
SECR[19:16]： INS_LEN	插入间隔长度控制位： 在配置随机执行周期/硬件随机指令安全机制后，处理器将根据该控制位所指示的值来控制每次插入空白间隔/随机指令的周期数。该控制位表示允许插入空白间隔/随机指令周期的最大值。处理器将在该最大值范围之内随机选择插入的周期数。
SECR[3:0]： INS_FREQ	插入间隔频度控制位： 在配置随机执行周期/硬件随机指令安全机制后，处理器将根据该控制位所指示的值来控制插入空白间隔/随机指令的频率。该控制位表示两次插入间允许执行指令数量的最大值。处理器将在该最大值范围之内随机选择两次插入间间隔的指令数。
相关信号：	
pad_seu_rand_num[31:0]	随机数信号： 指示真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

图表 7-5 安全机制信息

随机执行周期安全机制可将 S802 处理器的指令执行周期动态随机化。当该安全机制被使能时，安全扩展单元根据系统输入的真随机值，在指令执行过程中随机插入空白间隔，确保指令执行周期的随机性。处理器使能随机指令周期机制后，执行周期随机化，与操作数无关；不同输入下的功耗波形在时域上将无法对应，因而无法采用叠加统计的方式加以分析。



图表 7-6 随机执行周期示意图

要使能随机执行周期，首先需要设置插入间隔长度控制位 INS_LEN（SECR[19:16]）和插入间隔频度控制位 INS_FREQ（SECR[3:0]）。两者对一个随机数进行选择，该值作为对应计数器的初始值。其中，插入间隔长度控制位控制插入空白间隔的持续周期数。其值越大，插入的空白间隔越长，执行周期随机化越明显，对性能影响越大。插入间隔频度控制位控制插入空白间隔的频度，即两次插入空白间隔间的指令数。其值越小，插入空白间隔越频繁，执行周期随机化越明显，对性能影响越大。插入的空白间隔最短持续 1 个周期。两次插入空白间隔间最少间隔一条指令。

该机制需要系统通过接口信号（pad_seu_rand_num[31:0]）输入真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

该安全机制通过安全扩展单元控制寄存器的 RBI 位（SECR[8]）控制。RBI 位置 1 将开启随机执行周期。RBI 位置 0 将关闭随机执行周期。

调试模式下随机执行周期被关闭，且无法被使能。

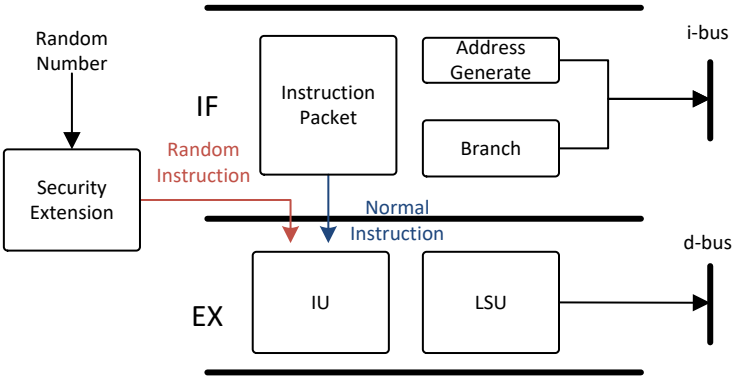
7.4. 硬件随机指令

抗攻击类型：
简单功耗分析、差分功耗分析
可配置性：

硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[4]：HRII	使能位：硬件随机指令使能
SECR[19:16]： INS_LEN	插入间隔长度控制位： 在配置随机执行周期/硬件随机指令安全机制后，处理器将根据该控制位所指示的值来控制每次插入空白间隔/随机指令的周期数。该控制位表示允许插入空白间隔/随机指令周期的最大值。处理器将在该最大值范围之内随机选择插入的周期数。
SECR[3:0]： INS_FREQ	插入间隔频度控制位： 在配置随机执行周期/硬件随机指令安全机制后，处理器将根据该控制位所指示的值来控制插入空白间隔/随机指令的频率。该控制位表示两次插入间允许执行指令数量的最大值。处理器将在该最大值范围之内随机选择两次插入间间隔的指令数。
相关信号：	
pad_seu_rand_num[31:0]	随机数信号： 指示真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

图表 7-7 安全机制信息

S802 安全扩展单元支持硬件随机指令插入功能。硬件随机指令安全机制根据输入的真随机数，在指令编码域内产生合法编码的随机指令。处理器根据当前指令流和安全扩展单元的请求，选择执行随机指令。随机指令通过指令执行单元产生与正常指令相同的功耗，增加简单功耗分析的难度。除此之外，随机指令可以扰乱正常指令的时序，抵御差分功耗分析攻击。硬件随机指令插入无需软件干预，不破坏正常指令流现场。随机指令插入的频度和强度都可以通过软件进行控制。



图表 7-8 硬件随机指令示意图

要使能硬件随机指令，首先需要设置插入间隔长度控制位 INS_LEN（SECR[19:16]）和插入间隔频度控制位 INS_FREQ（SECR[3:0]）。两者对一个随机数进行选择，该值作为对应计数器的初始值。其中，插入间隔长度控制位控制插入随机指令的数量。其值越大，插入的随机指令越多，对性能影响越大。插入间隔频度控制位控制插入随机指令的频度，即两次插入随机指令间的指令数。其值越小，插入随机指令越频繁，对性能影响越大。每次最少插入一条指令。两次插入随机指令间最少间隔一条正常指令，且特定周期和序列的指令之间无法插入随机指令。

该机制需要系统通过接口信号（pad_seu_rand_num[31:0]）输入真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

该安全机制通过安全扩展单元控制寄存器的 HRII 位（SECR[4]）控制。HRII 位置 1 将开启硬件随机指令。HRII 位置 0 将关闭硬件随机指令。

调试模式下硬件随机指令被关闭，且无法被使能。

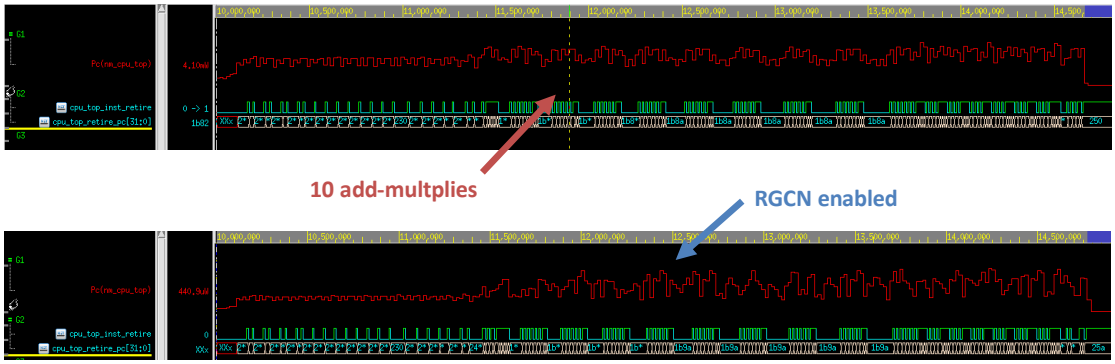
7.5. 随机时钟噪声源

抗攻击类型：	
简单功耗分析	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[9]：RGCN	使能位：随机时钟噪声源使能

相关信号：	
pad_seu_rand_num[31:0]	随机数信号： 指示真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

图表 7-9 安全机制信息

S802 安全扩展单元支持随机时钟噪声源功能。该安全机制可以根据输入的真随机数，利用处理器内部的门控时钟网络，产生与正常指令执行相当量级的随机功耗噪声。随机时钟噪声源不仅没有增加任何面积成本，而且不对处理器性能产生任何负面影响，是一个有效的无成本的功耗噪声源。



图表 7-10 随机时钟噪声源示意图

该机制需要系统通过接口信号（pad_seu_rand_num[31:0]）输入真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

该安全机制通过安全扩展单元控制寄存器的 RGCN 位（SECR[9]）控制。RGCN 位置 1 将开启随机时钟噪声源。RGCN 位置 0 将关闭随机时钟噪声源。

调试模式下随机时钟噪声源被关闭，且无法被使能。

7.6. 通用寄存器校验

抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元，需选择一种校验算法）
软件配置类型：	无

相关寄存器：	
无	
相关信号：	
无	

图表 7-11 安全机制信息

S802 的安全扩展单元对所有通用寄存器进行校验，确保被错误注入攻击篡改后，错误值可以被检测出来。通用寄存器校验逻辑捕捉正常回写时的校验值，将其与读取通用寄存器时的校验值比较，保证通用寄存器内存放的值不被正常回写以外的其它非法操作所修改。

通用寄存器校验所采用的校验算法是可配置的。在选择配置该安全机制时，必须选择一种校验算法。有关校验算法请参考 7.9 可配置校验算法。

当硬件配置通用寄存器校验后，校验随时进行，无法通过软件关闭。

调试模式下安全机制不会响应校验失败。

7.7. 控制寄存器校验

抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元，需选择一种校验算法）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
无	

图表 7-12 安全机制信息

S802 的安全扩展单元对所有控制寄存器进行校验，确保被错误注入攻击篡改后，错误值可以被检测出来。控制寄存器校验逻辑捕捉正常写入时的校验值，将其与读取控制寄存器时的校验值比较，保证控制寄存器内存放的值不被正常回写以外的其它非法操作所修改。

控制寄存器校验所采用的校验算法是可配置的。在选择配置该安全机制时，必须选择一种校验算法。有关校验算法请参考 7.9 可配置校验算法。

当硬件配置控制寄存器校验后，校验随时进行，无法通过软件关闭。

调试模式下安全机制不会响应校验失败。

7.8. 流水线校验

抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元，需选择一种校验算法）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
无	

图表 7-13 安全机制信息

S802 的安全扩展单元对所有流水线寄存器进行校验，确保被错误注入攻击篡改后，错误值可以被检测出来。流水线寄存器校验逻辑捕捉正常写入时的校验值，将其与读取流水线寄存器时的校验值比较，保证流水线寄存器内存放的值不被正常回写以外的其它非法操作所修改。

流水线校验所采用的校验算法是可配置的。在选择配置该安全机制时，必须选择一种校验算法。有关校验算法请参考 7.9 可配置校验算法。

当硬件配置流水线校验后，校验随时进行，无法通过软件关闭。

调试模式下安全机制不会响应校验失败。

7.9. 可配置校验算法

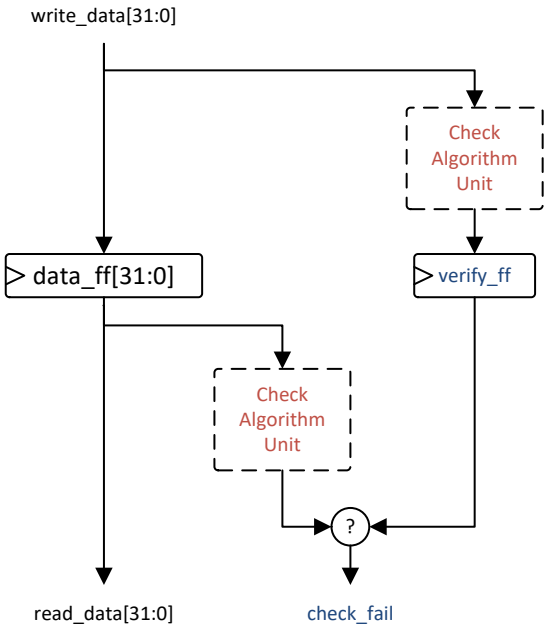
抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	需配置安全扩展单元，需选择一种校验安全机制
软件配置类型：	无
相关寄存器：	
无	

相关信号：	
无	

图表 7-14 安全机制信息

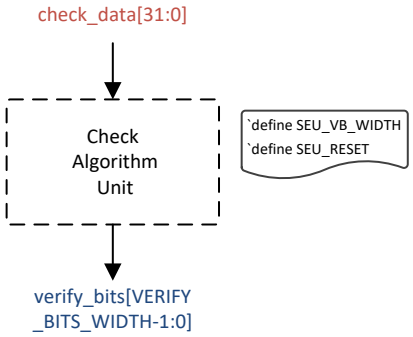
S802 安全扩展单元的校验安全机制所采用的校验算法可以被灵活配置，以适应不同的安全需求与指标。安全扩展单元预置了两种常见的校验算法。用户可根据需求选择奇偶校验算法或汉明校验算法，也可以选择自定义校验算法。可配置校验算法影响通用寄存器校验、控制寄存器校验和流水线校验安全机制。

预置的两种校验算法具有不同的校验强度和成本。奇偶校验消耗较少资源，可以检测 32 位寄存器中任意一位被篡改的情况。汉明校验消耗较多资源，可以检测 32 位寄存器中任意两位被篡改的情况。



图表 7-15 校验算法实例化示意图

用户可以选择自定义校验算法。该校验算法需在当前周期内，根据被校验的数据计算出校验值。自定义校验算法单元在被校验寄存器的读写逻辑处被多次实例化，因此其时序需满足 S802 处理器核内时序要求。请联系相关人员获取自定义校验算法的接口标准。



图表 7-16 自定义校验算法接口示意图

可配置校验算法仅在通用寄存器校验、控制寄存器校验或流水线校验安全机制配置时有效。

7.10. 程序计数器校验

抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
无	

图表 7-17 安全机制信息

S802 的安全扩展单元对程序计数器进行校验，确保被错误注入攻击篡改后，错误值可以被检测出来。该安全机制利用流水线不同阶段的程序计数器互为校验，保证任意一个程序计数器内存放的值不被正常写入以外的其它非法操作所修改。程序计数器校验无需校验算法单元，不受 7.9 可配置校验算法影响。

当硬件配置程序计数器校验后，校验随时进行，无法通过软件关闭。

调试模式下安全机制不会响应校验失败。

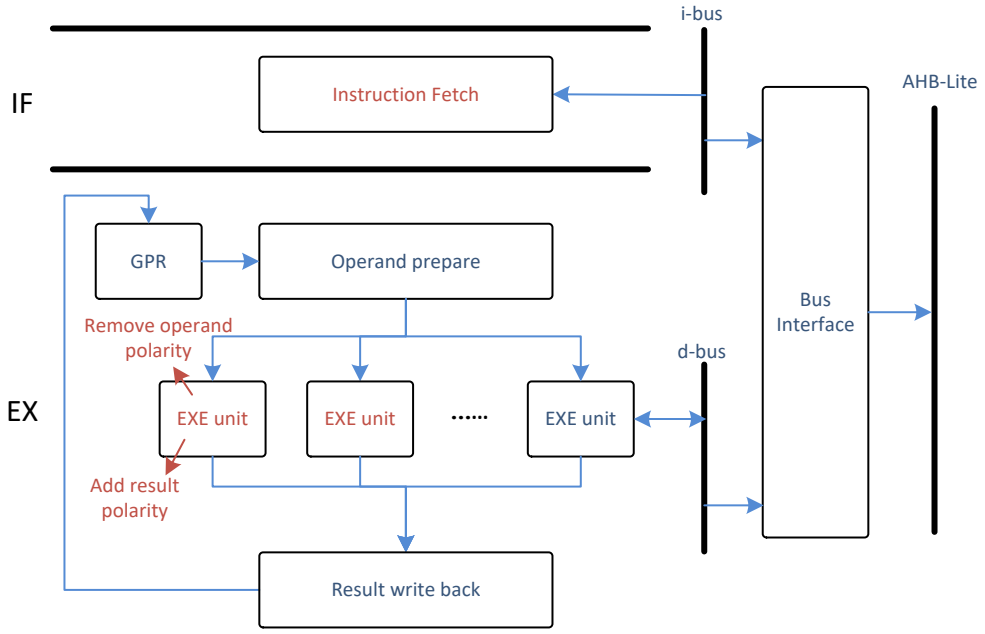
7.11. 数据通路极性翻转

抗攻击类型：	
差分功耗分析	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[12]：POL	使能位：数据通路极性翻转使能
SECR[24:20]： POL_FREQ	插入极性间隔长度控制位： 在配置数据通路极性翻转机制后，处理器根据该控制位所指示的值来控制每次对处理器内部数据通路加入极性的频率。该控制位表示两次加入极性之间的最大间隔周期数。处理器将在该最大值范围之内随机选择两次插入极性的间隔周期数。
相关信号：	
pad_seu_rand_num[31:0]	随机数信号： 指示真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。
biu_pad_haddr_pol	总线读写传输地址极性翻转信号： 标记该次读写传输地址是否进行了极性翻转。（仅在配置总线极性反转位时有效）
biu_pad_hwdata_pol	总线写数据极性翻转信号： 标记该次写数据是否进行了极性翻转。（仅在配置总线极性反转位时有效）
pad_biu_hrdata_pol	总线读数据极性翻转信号： 标记该次读数据是否进行了极性翻转。（仅在配置总线极性反转位时有效）

iahbl_pad_haddr_pol	指令总线读写传输地址极性翻转信号： 标记该次读写传输地址是否进行了极性翻转。（仅在配置总线极性反转位时有效）
iahbl_pad_hwdata_pol	指令总线写数据极性翻转信号： 标记该次写数据是否进行了极性翻转。（仅在配置总线极性反转位时有效）
pad_iahbl_hrdata_pol	指令总线读数据极性翻转信号： 标记该次读数据是否进行了极性翻转。（仅在配置总线极性反转位时有效）
dahbl_pad_haddr_pol	数据总线读写传输地址极性翻转信号： 标记该次读写传输地址是否进行了极性翻转。（仅在配置总线极性反转位时有效）
dahbl_pad_hwdata_pol	数据总线写数据极性翻转信号： 标记该次写数据是否进行了极性翻转。（仅在配置总线极性反转位时有效）
pad_dahbl_hrdata_pol	数据总线读数据极性翻转信号： 标记该次读数据是否进行了极性翻转。（仅在配置总线极性反转位时有效）

图表 7-18 安全机制信息

S802 支持数据通路极性翻转安全机制。在该安全机制下，S802 处理器内部的所有数据通路都具有极性，包括指令操作数、通用寄存器、内部数据总线等。S802 从通用寄存器中获得具有极性的数据，作为操作数进行准备，发送至执行单元。执行单元将操作数的极性解除后参与运算，并按照输入的真随机数对结果的极性进行翻转。当配置数据通路极性翻转后，S802 总线接口单元可以发送具有极性的地址和数据给外部总线，并从外部总线接收具有极性的数据。传输地址极性由 S802 核内数据通路随传输地址一起发送到总线；写数据极性由 S802 核内数据通路随写数据一起发送到总线；读数据极性由总线从设备随读数据产生，总线接口单元接收读数据和相关极性位并将其传输至 S802 核内。如果配置独立指令总线和数据总线，这两个总线接口的地址和数据也需要附带极性。



图表 7-19 数据通路极性翻转示意图

该机制需要系统通过接口信号（pad_seu_rand_num[31:0]）输入真随机数。真随机数需要在该安全机制使能前开始指示，至安全机制关闭后为止。安全机制使能时，真随机数在每个周期都应当产生变化。

该安全机制通过安全扩展单元控制寄存器的 POL 位（SECR[12]）控制。POL 位置 1 将开启数据通路极性翻转。数据通路极性翻转开启后，指令执行新生成的运算结果将具有随机的极性翻转信息。极性翻转的频率根据 SECR[24:20]所控制的粒度产生，具体的翻转频率控制机制为：若 SECR[24:20]的值被设置为 M，则处理器内部的数据通路会每隔 N 个周期产生一次极性（即 pol 位为 1），N 为随机数，且 1≤N≤M。

下表为推荐用户配置的粒度控制值。

SECR[24:20]初始化	数据通路极性产生周期 N（随机数）
5’b00001	1≤N≤2
5’b00011	1≤N≤4
5’b00111	1≤N≤8
5’b01111	1≤N≤16
5’b11111	1≤N≤32

需要注意的是，外部输入给处理器的极性处理器无法自行控制，因此，如果需要控制外部输入的极性翻转粒度，需要在 SOC 层面进行控制。

POL 位置 0 将关闭数据通路极性翻转。数据通路极性翻转关闭后，指令执行新生成的运算结果将不具有极性翻转信息。

调试模式下数据通路极性翻转被关闭，且无法被使能。

7.12. 关键寄存器互补备份

抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
无	

图表 7-20 安全机制信息

S802 的安全扩展单元对关键寄存器进行互补备份，防止其被错误注入攻击篡改。备份寄存器内存放关键寄存器的互补值，与原值互为校验。进行互补备份的关键寄存器包括处理器状态寄存器内的超级用户模式设置位（PSR.S）、异常保留处理器状态寄存器内的超级用户模式设置位（EPSR.S）和高速缓存配置寄存器内的内存保护设置位（CCR.MP）。

当硬件配置程序计数器校验后，校验随时进行，无法通过软件关闭。

调试模式下安全机制不会响应校验失败。

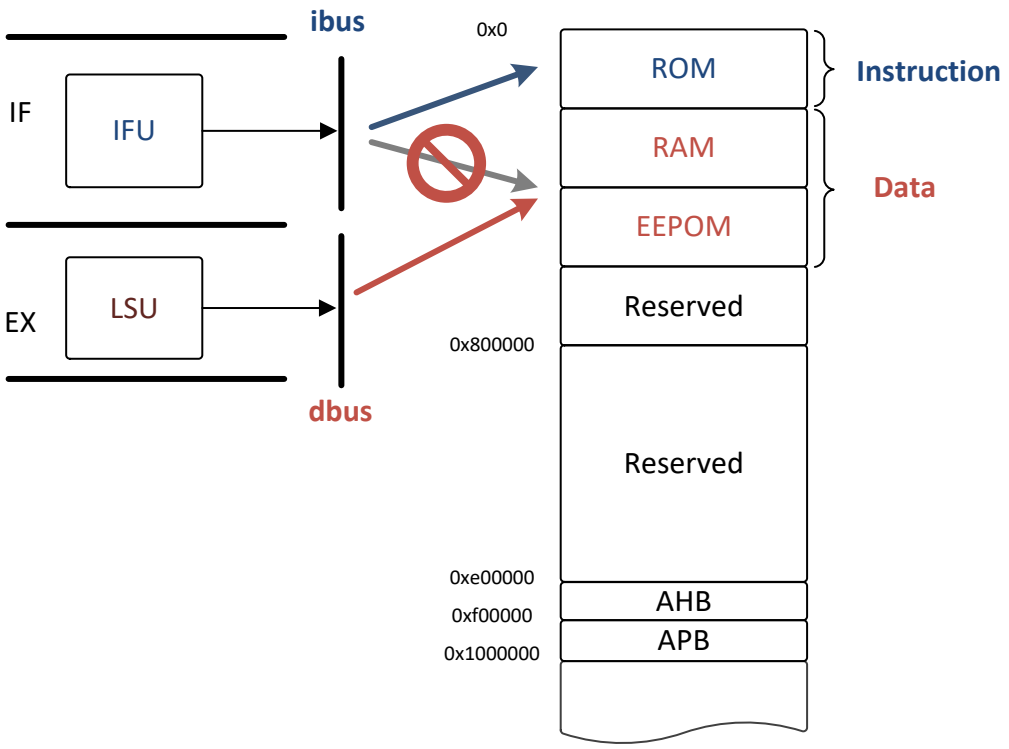
7.13. 保护区可执行检查

抗攻击类型：	
缓冲区溢出攻击	
可配置性：	
硬件配置类型：	随内存管理单元配置（需配置内存管理单元）
软件配置类型：	可通过可高缓和访问权限配置寄存器（CAPR）配置
相关寄存器：	

CAPR[7:0]	不可执行位属性配置位： 用于设置对应保护区的可执行属性。当设置为 1 时，该保护区内的指令不可以被执行。反之，保护区内的指令可以被执行。 保护区默认为可执行属性。 该寄存器的配置请参考 S802 用户手册。
相关信号：	
无	

图表 7-21 安全机制信息

S802 可以通过内存保护单元（MPU）对保护区的可执行属性进行检查，阻止任何来源于非指令区的非法取值请求。当内存保护单元将某个保护区配置为不可执行后，处理器核内指令总线对该区域的访问请求将被阻止，并由内存保护单元返回访问禁止（Access Error）。



图表 7-22 保护区可执行检查示意图

保护区可执行检查随内存管理单元（MPU）硬件配置。该安全机制通过配置保护区属性设置。不可执行位属性配置位位于可高缓和访问权限配置寄存器的最低有效位（CAPR[7:0]）。当某个保护区对应的不可执行位属性配置位为 1 时，该区域不可执行。保

护区默认为可执行。内存管理单元的保护区配置方法请参考 S802 用户手册。

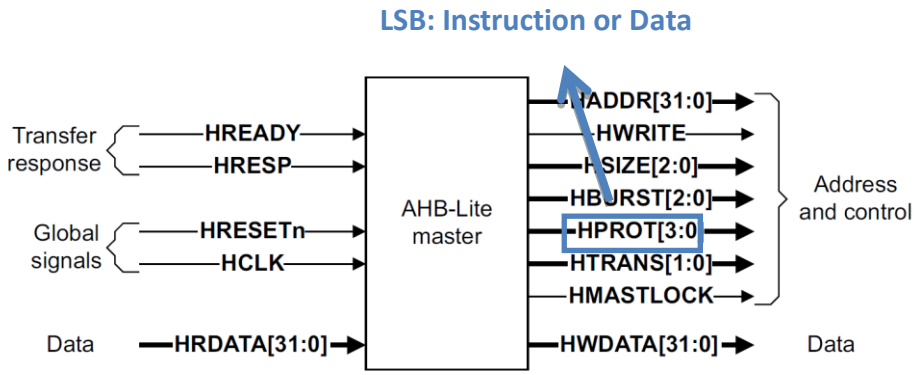
任何对不可执行属性的保护区的取指令请求将会得到访问禁止（Access Error），而不会产生安全违反。

7.14. 显式内存访问属性

抗攻击类型：	
缓冲区溢出攻击	
可配置性：	
硬件配置类型：	无（S802基本配置）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
biu_pad_hprot[0]	总线接口单元内存访问属性信号： 指示总线单元的内存访问请求属性。该信号为 0 表示当前内存访问请求来自于处理器内数据总线，1 表示请求来自于指令总线。 该信号的配置请参考 S802 用户手册。
iahbl_pad_hprot[0]	指令总线接口单元内存访问属性信号： 指示总线单元的内存访问请求属性。该信号为 0 表示当前内存访问请求来自于处理器内数据总线，1 表示请求来自于指令总线。 该信号的配置请参考 S802 用户手册。
dahbl_pad_hprot[0]	数据总线接口单元内存访问属性信号： 指示总线单元的内存访问请求属性。该信号为 0 表示当前内存访问请求来自于处理器内数据总线，1 表示请求来自于指令总线。 该信号的配置请参考 S802 用户手册。

图表 7-23 安全机制信息

S802 处理器的任何内存访问都附带访问属性。当 S802 发起内存访问请求时，该安全机制将指示当前内存访问的属性。指示信号复用总线接口信号 hprot 的最低位：1 表示当前内存访问请求来自于指令总线；0 表示当前内存访问请求来自于数据总线。



图表 7-24 显示内存访问属性示意图

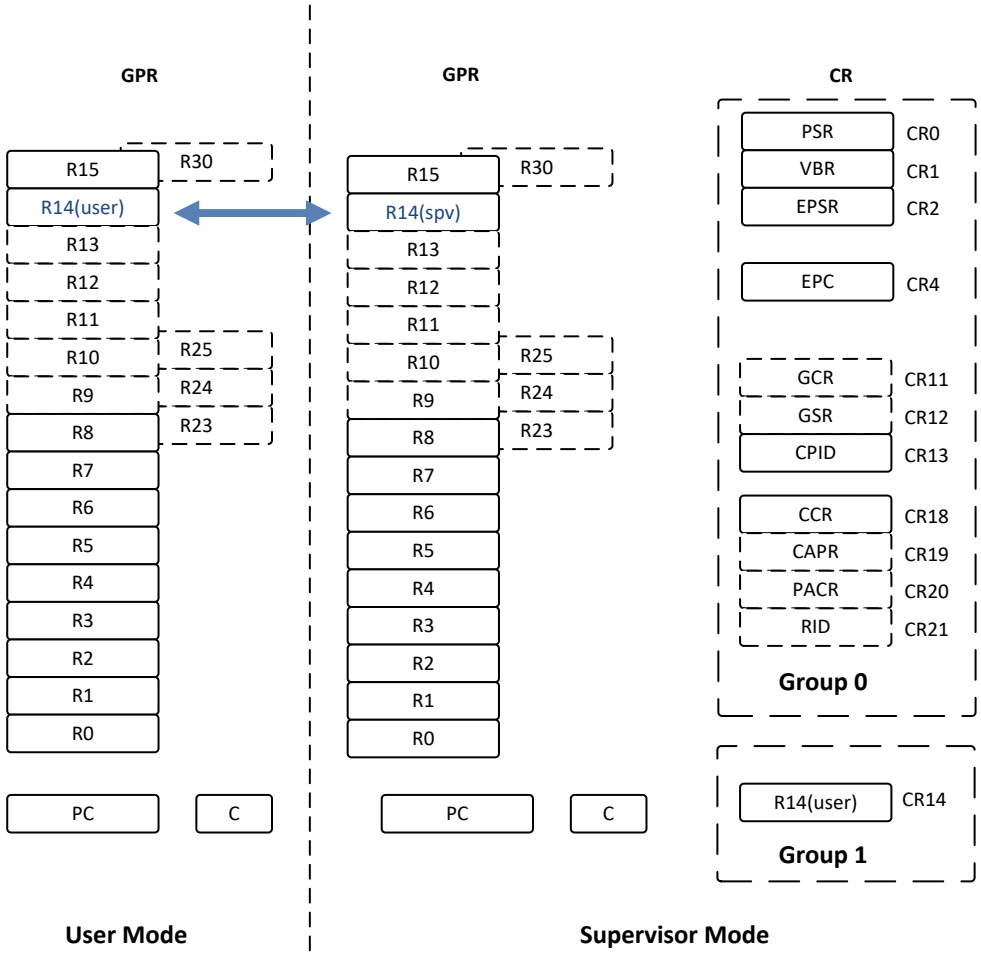
显示内存访问属性作为 S802 的基本配置，随对应的总线接口信号 HPROT 指示。总线接口信号请参考 S802 用户手册。

7.15. 独立堆栈指针

抗攻击类型：	
缓冲区溢出攻击	
可配置性：	
硬件配置类型：	无（S802基本配置）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
无	

图表 7-25 安全机制信息

CSKY V2 编程模型中，普通用户模式和超级用户模式的堆栈指针是分离的。任何对堆栈指针寄存器（R14）的访问，将会根据当前处理器所处的模式指向对应的堆栈指针寄存器。这将避免超级用户模式下的堆栈被普通用户模式程序访问。CSKY V2 编程模型还为超级用户模式提供了访问普通用户模式堆栈指针寄存器的接口。堆栈指针寄存器的编程模型请参考 CSKY V2 指令集用户手册。



图表 7-26 显示内存访问属性示意图

独立堆栈指针是 S802 的基本配置。

7.16. 外部通用寄存器复位

抗攻击类型：	
攻击响应	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	无
相关寄存器：	
无	

相关信号：	
pad_cpu_gpr_rst_b	通用寄存器复位信号： 用于指示将 S802 处理器所有通用寄存器同步复位。该信号低电平有效。

图表 7-27 安全机制信息

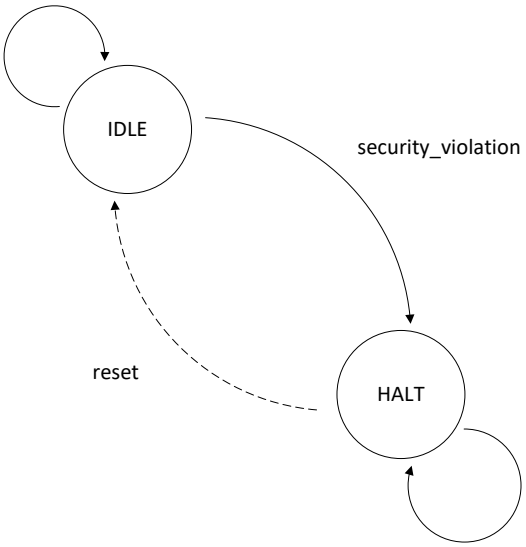
S802 支持外部通用寄存器复位。该同步复位信号可以将处理器内部通用寄存器立刻复位，确保现场被快速清除。外部通用寄存器复位可硬件独立配置，通过 pad_cpu_gpr_rst_b 指示，低电平有效。

7.17. 安全违反检测与上报

抗攻击类型：	
攻击响应	
可配置性：	
硬件配置类型：	随安全扩展单元配置
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
iu_pad_security_violation	安全违反信号： 指示处理器处于安全违反状态。该信号为 1 表示处理器安全违反，0 表示没有发生安全违反。

图表 7-28 安全机制信息

当 S802 处理器产生安全违反后，安全扩展单元将处理器置于安全违反状态。在该状态下，处理器被挂起，所有指令执行停止。当处理器处于安全违反状态时，安全违反信号 iu_pad_security_violation 为 1。只有复位可以使处理器退出安全违反状态。



图表 7-29 安全违反状态示意图

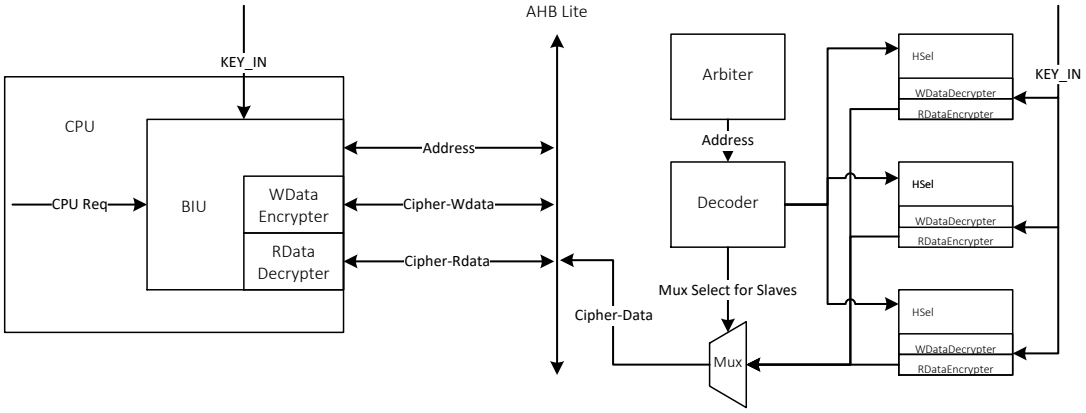
安全违反检测与上报是 S802 安全扩展单元的基本机制，随安全扩展单元配置。

7.18. 总线数据加扰

抗攻击类型：	
简单功耗分析	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	无
相关寄存器：	
无	
相关信号：	
pad_biu_beu_key0[23:0]	总线加扰机制密钥信号
pad_biu_beu_key1[23:0]	总线加扰机制密钥信号

图表 7-30 安全机制信息

总线上的数据使用 pad_biu_beu_key0 以及 pad_biu_beu_key1 进行加扰，防止总线信号被直接分析，从而窃取程序流指令码和运算过程数据。数据在总线上传输的是加扰过的密文。



图表 7-31 总线数据加扰示意图

加扰过程使用的密钥通过信号接口输入至 S802 处理器，分为 pad_biu_beu_key0 和 pad_biu_beu_key1 两个子密钥。

用户可选择自定义加扰算法，该加扰算法需在当前周期内完成。自定义加扰算法单元在主设备和从设备两端实例化，因此其时序需满足 S802 处理器核内时序要求。请联系相关人员获取自定义加密算法的接口标准。

如果配置独立的指令总线和数据总线，该指令/数据总线单元也需要对总线数据进行加/解扰。

7.19. 总线数据奇偶校验

抗攻击类型：	
错误注入	
可配置性：	
硬件配置类型：	可独立硬件配置（需配置安全扩展单元）
软件配置类型：	可通过安全扩展单元控制寄存器配置
相关寄存器：	
SECR[7]：BP	使能位：总线数据奇偶校验使能
相关信号：	
biu_pad_hwdata_par	总线写数据奇偶校验信号： 记录当前写数据的奇偶校验位。（仅在配置总线奇偶校验机制时有效）

pad_biu_hrdata_par	总线读数据奇偶校验信号： 记录当前读数据的奇偶校验位。（仅在配置总线奇偶校验机制时有效）
iahbl_pad_hwdata_par	指令总线写数据奇偶校验信号： 记录当前写数据的奇偶校验位。（仅在配置总线奇偶校验机制时有效）
pad_iahbl_hrdata_par	指令总线读数据奇偶校验信号： 记录当前读数据的奇偶校验位。（仅在配置总线奇偶校验机制时有效）
dahbl_pad_hwdata_par	数据总线写数据奇偶校验信号： 记录当前写数据的奇偶校验位。（仅在配置总线奇偶校验机制时有效）
pad_dahbl_hrdata_par	数据总线读数据奇偶校验信号： 记录当前读数据的奇偶校验位。（仅在配置总线奇偶校验机制时有效）

图表 7-32 安全机制信息

S802 的安全扩展支持总线数据奇偶校验，通过对总线的读写数据进行奇偶校验，确保数据不被外部攻击篡改。当配置总线数据奇偶校验机制后，总线的读写数据将各配置一个奇偶校验位，指示发起端奇偶校验运算的结果。S802 在接收端对传输数据重新校验，并将校验结果与随总线传输过来的原始奇偶校验位进行对比，以检测总线数据是否遭受错误注入攻击。该安全机制采用奇偶校验算法，可以检测总线数据任意一位被修改的情况。

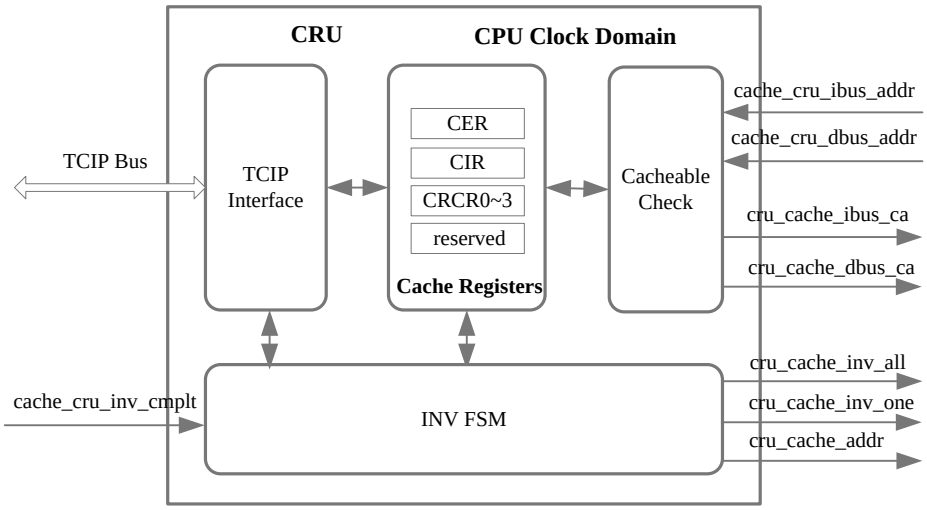
该安全机制通过安全扩展单元控制寄存器的 BP 位（SECR[7]）控制。BP 位置 1 将开启总线数据奇偶校验。BP 位置 0 将关闭总线数据奇偶校验，此时奇偶校验失败将不会引起安全违反。

调试模式下安全机制不会响应校验失败。

8. 片上高速缓存

8.1. 高速缓存简介

S802 CPU 提供硬件可配置的高速缓存器（Cache）。Cache 控制寄存器单元（CRU）为 Cache 提供了一组设置寄存器，包括 Cache 的使能、缓存行的无效和高速缓存区的配置。



图表 8-1 Cache 控制寄存器单元结构图

8.2. 相关系统控制寄存器

CRU 提供一组 32-bit 的寄存器，各个寄存器地址空间如图表 7-2 所示。

地址	名称	类型	初始值	描述
0xE000F000	CER	读/写	0x00000000	高速缓存使能寄存器
0xE000F004	CIR	读/写	0x00000000	高速缓存无效寄存器
0xE000F008	CRCR0	读/写	0x00000000	0号可高缓区配置寄存器
0xE000F00C	CRCR1	读/写	0x00000000	1号可高缓区配置寄存器（配置2个或2个以上可高缓区）
0xE000F010	CRCR2	读/写	0x00000000	2号可高缓区配置寄存器（配置3个或3个以上可高缓区）
0xE000F014	CRCR3	读/写	0x00000000	3号可高缓区配置寄存器（配置4个可高缓区）

0xE000F018~ 0xE000FFFF	-	-	-	保留

图表 8-2 Cache 控制寄存器定义

8.2.1. 高速缓存使能寄存器(CER)

	31	2	1	0
	Reserved		CFIG	EN
Reset:	-		0	0

图表 8-3 高速缓存使能寄存器

EN-Cache 使能位：

当 EN 为 0 时，Cache 处于关闭状态。

当 EN 为 1 时，Cache 处于工作状态。

CIFG-Cache 属性配置位：

0：指令与数据 Cache；

1：指令 Cache。

8.2.2. 高速缓存无效寄存器(CIR)

	31	4	3	2	1	0
	INV_ADDR		RESERVED		INV_ONE	INV_ALL
Reset:	0		-		0	0

图表 8-4 高速缓存无效寄存器

INV_ALL-整个 Cache 无效设置位：

当 INV_ALL 为 1 时，无效 Cache 中所有缓存行。

INV_ONE-单条缓存行无效设置位：

当 INV_ONE 为 1 时，无效选中的缓存行。

INV_ADDR-缓存行地址：

INV_ADDR 表征需要被无效的缓存行地址。

REV-保留位

8.2.3. 可高缓区配置寄存器 0~3（CRCR）

31 10	9 6	5 1	0
Base Address	REV	Size	EN
Reset: 0	-	0	0

图表 8-5 可高缓区配置寄存器

Base Address-可高缓区基地址：

可高缓区大小 1KB 到 4GB 可配置，该域指出了可高缓区地址的高位，例如设置页面大小为 8M，CRCR [22:10]必须为 0，不同大小的区各页面的具体要求见下表。

Size-保护区大小：

可高缓区大小可以通过公式：可高缓区大小＝2^（Size+1）计算得到。因此 Size 可设置的范围为 01001 到 11111，其它一些值都会造成不可预测的结果。

EN-可高缓区有效位：

当 EN 为 0 时，可高缓区无效；

当 EN 为 1 时，可高缓区有效。

Size	可高缓区大小	对基地址低位的要求
00000—01010	保留	—
01001	1KB	-
01010	2KB	CRCR.bit[10]=0
01011	4KB	CRCR.bit[11:10]=0
01100	8KB	CRCR.bit[12:10]=0
01101	16KB	CRCR.bit[13:10]=0
01110	32KB	CRCR.bit[14:10]=0
01111	64KB	CRCR.bit[15:10]=0
10000	128KB	CRCR.bit[16:10]=0
10001	256KB	CRCR.bit[17:10]=0
10010	512KB	CRCR.bit[18:10]=0
10011	1MB	CRCR.bit[19:10]=0
10100	2MB	CRCR.bit[20:10]=0
10101	4MB	CRCR.bit[21:10]=0

Size	可高缓区大小	对基地址低位的要求
10110	8MB	CRCR.bit[22:10]=0
10111	16MB	CRCR.bit[23:10]=0
11000	32MB	CRCR.bit[24:10]=0
11001	64MB	CRCR.bit[25:10]=0
11010	128MB	CRCR.bit[26:10]=0
11011	256MB	CRCR.bit[27:10]=0
11100	512MB	CRCR.bit[28:10]=0
11101	1GB	CRCR.bit[29:10]=0
11110	2GB	CRCR.bit[30:10]=0
11111	4GB	CRCR.bit[31:10]=0

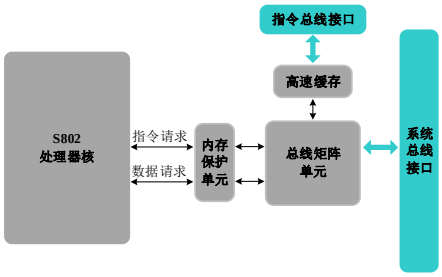
图表 8-6 可高缓区大小配置和其对基址要求

9. 总线矩阵与总线接口

9.1. 简介

S802 实现了多总线接口，分别包括系统总线、指令总线。指令总线可由用户根据实际的系统需要进行配置。

总线矩阵为处理器内部请求访问外部总线接口提供了互联功能。总线矩阵与 CPU 内部请求及总线接口的连接关系如图表 9-1 所示。总线矩阵根据内存访问的地址仲裁总线接口类型，将处理器内部访问分发到系统总线、指令总线上。



图表 9-1 S802 总线矩阵

处理器内部的取指访问和数据访问拥有相同的总线访问权限，可以访问所有总线接口。为了解决同一时钟周期取指访问和数据访问竞争同一总线接口的问题，总线矩阵也负责请求的优先级判断。当取指请求和数据请求竞争同一总线接口时，数据请求拥有更高的优先级。

S802 多总线接口的基本信息和可配置性如图表 9-2 所示。

图表 9-2 多总线接口的基本信息和可配置性

总线接口	可配置性（有/无）	总线协议	时序方式
系统总线	可配置	AHB	寄存器输出
		AHB-Lite	直接输出（推荐）
指令总线	可配置	AHB-Lite	寄存器输出
			直接输出（推荐）

另外，S802 通过提供一组接口信号（pad_bmu_iahbl_base 和 pad_bmu_iahbl_mask），支持指令总线基地址和空间大小硬件可配置。其中，pad_bmu_iahbl_base 指定了指令总线的基地址；pad_bmu_iahbl_mask 指定了不同地址空间下对地址对齐的需求。指令总线的地址空间 1MB 到 4GB 可配置，例如设置地址空间大小为 8M，pad_bmu_iahbl_base[2:0]必须为 3’b0，pad_bmu_iahbl_mask[11:0]必须为 12’b1111 1111 1000，不同大小的地址空间具体要求见下表。

图表 9-3 指令总线对基地址和地址对齐的要求

地址空间大小	对 pad_bmu_iahbl_base 的要求	对 pad_bmu_iahbl_mask 的要求
1MB	没有要求	bit[11:0]=12'b1111 1111 1111
2MB	bit[0] =0	bit[11:0]=12'b1111 1111 1110
4MB	bit[1:0] =2'b0	bit[11:0]=12'b1111 1111 1100
8MB	bit[2:0] =3'b0	bit[11:0]=12'b1111 1111 1000
16MB	bit[3:0] =4'b0	bit[11:0]=12'b1111 1111 0000
32MB	bit[4:0] =5'b0	bit[11:0]=12'b1111 1110 0000
64MB	bit[5:0] =6'b0	bit[11:0]=12'b1111 1100 0000
128MB	bit[6:0] =7'b0	bit[11:0]=12'b1111 1000 0000
256MB	bit[7:0] =8'b0	bit[11:0]=12'b1111 0000 0000
512MB	bit[8:0] =9'b0	bit[11:0]=12'b1110 0000 0000
1GB	bit[9:0] =10'b0	bit[11:0]=12'b1100 0000 0000
2GB	bit[10:0] =11'b0	bit[11:0]=12'b1000 0000 0000
4GB	bit[11:0] =12'b0	bit[11:0]=12'b0000 0000 0000

9.2. 系统总线接口

9.2.1.特点

S802 的系统总线接口可以配置为支持 AMBA2.0 AHB 协议（此时请参考 AMBA 2.0 规格说明—AMBA™ Specification Rev 2.0），也可以配置为支持 AMBA3.0 AHB-Lite 协议（此时请参考 AMBA 3.0 规格说明—AMBA3 AHB-Lite Protocal Specification Rev 1.0）。

系统总线接口的基本特点包括：

- 支持 AMBA2.0 AHB 或 AMBA3.0 AHB-Lite 总线协议，可由用户配置；
- 支持配置为寄存器输出方式（Flop-out）或直接输出（non-Flop-out）；
- 当配置为寄存器输出方式（Flop-out）时支持同步工作模式，支持多种处理器和系统时钟的频率比；当配置为直接输出（non-Flop-out）时，处理器和系统时钟频率比必须为 1：1；
- 当配置为寄存器输出方式（Flop-out）时支持系统动态变频（动态调整处理器和系统时钟的频率比）。

9.2.2.协议内容

9.2.2.1. 支持传输类型

考虑到 S802 的应用领域及成本，系统总线接口只实现了 AHB/AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 只支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 只支持 IDLE 和 NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

在 AHB 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 支持 IDLE、NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

9.2.2.2. 支持响应类型

在 AHB 及 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

9.2.3. 不同总线响应下的行为

图表 9-4 列出了总线上出现不同总线响应时 CPU 的行为。

图表 9-4 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

9.2.4. AHB 协议的接口信号

图表 9-5 AHB 协议接口信号

信号名	I/O	Reset	定义

biu_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
biu_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
biu_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 S802 仅支持 SINGLE 突发传输。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 00：IDLE； 10：NONSEQ； S802 支持 IDLE、NONSEQ 传输类型。
biu_pad_hwrite	O	0	读写表示信号： 1：指示是写总线传输； 0：指示是读总线传输。

biu_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
biu_pad_hbusreq	O	0	总线请求信号： 指示 CPU 请求总线的使用权。
pad_biu_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。
pad_biu_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_biu_hgrant	I	-	总线占用指示信号： 有效时指示 CPU 当前已占用总线。
pad_biu_hresp[1:0]	I	-	传输应答信号： 00：OKAY； 01：ERROR； 10：RETRY； 11：SPLIT。 S802 仅支持 OKAY 和 ERROR 响应类型。

9.2.5. AHB-Lite 协议的接口信号

图表 9-6 AHB-Lite 协议接口信号

信号名	I/O	Reset	定义
biu_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
biu_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
biu_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 S802 仅支持 SINGLE 突发传输。
biu_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。
biu_pad_htrans[1:0]	O	00	传输类型表示信号： 00：IDLE； 01：BUSY； 10：NONSEQ； 11：SEQ。 S802 仅支持 IDLE 和 NONSEQ 传输类型。

biu_pad_hwrite	O	0	读写表示信号： 1：指示是写总线传输； 0：指示是读总线传输。
biu_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
pad_biu_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。
pad_biu_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_biu_hresp[1:0]	I	-	传输应答信号： 00：OKAY； 01：ERROR； 10：RETRY； 11：SPLIT。 S802 仅支持 OKAY 和 ERROR 响应类型。

9.3. 指令总线接口

9.3.1.特点

S802 的指令总线只支持 AMBA3.0 AHB-LITE 协议，可参考 AMBA 3.0 规格说明-AMBA3 AHB-Lite Protocal Specification Rev 1.0。

指令总线接口的基本特点有：

- 与 AMBA3.0 AHB_LITE 总线协议兼容；
- 支持寄存器输出（Flop-out）和直接输出（Non-Flop-out）两种方式，可由用户配置。
- 在寄存器输出方式下，支持多种 CPU 和系统时钟的频率比；
- 在寄存器输出方式下，支持系统动态变频（调整 CPU 和系统时钟的频率比）。

9.3.2.协议内容

9.3.2.1. 支持传输类型

考虑到 S802 的应用领域及成本，指令总线接口只实现了 AHB-Lite 协议中的部分内容。

在 AHB-Lite 协议下，作为主设备，总线接口支持的传输类型为：

- HBURST 只支持 SINGLE 传输，其它突发类型均不支持；
- HTRANS 只支持 IDLE 和 NONSEQ，其它传输类型均不支持；
- HSIZE 支持字、字节和半字传输，其它传输大小不支持；
- HWRITE 支持读和写操作。

9.3.2.2. 支持响应类型

在 AHB-Lite 协议下，总线接口接受从设备的响应类型为：

- HREADY 支持 Ready 和 Not Ready；
- HRESP 支持 OKAY 和 ERROR，其它响应类型不支持。

9.3.3. 不同总线响应下的行为

图表 9-7 列出了总线上出现不同总线响应时 CPU 的行为。

图表 9-7 总线异常处理

HREADY	HRESP	结果
不关心	ERROR	访问错误，总线结束传输并处理访问错误。
High	OKEY	传输结束。
Low	OKEY	插入等待状态。

9.3.4. 指令总线接口信号

图表 9-8 指令总线接口信号

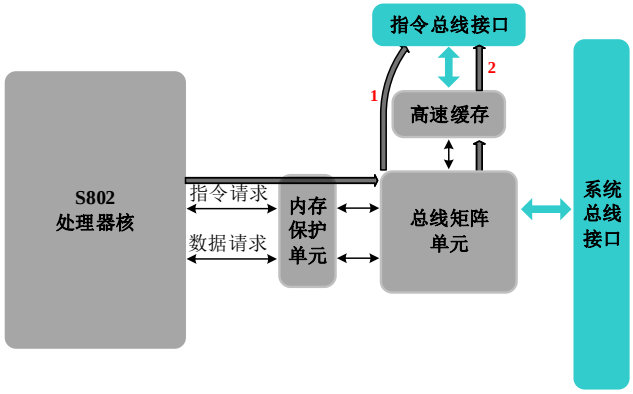
信号名	I/O	Reset	定义
iahbl_pad_haddr[31:0]	O	-	地址总线： 32 位地址总线。
iahbl_pad_hburst[2:0]	O	-	突发传输指示信号： 000：SINGLE； 001：INCR； 010：WRAP4。 S802 仅支持 SINGLE 突发传输。
iahbl_pad_hprot[3:0]	O	-	保护控制信号： ***0：取指令； ***1：数据访问； **0*：用户访问； **1*：超级用户访问； *0**：Not bufferable； *1**：bufferable； 0***：Not cacheable； 1***：cacheable。
iahbl_pad_hsize[2:0]	O	-	传输宽度指示信号： 000：byte； 001：halfword； 010：word。

iahbl_pad_htrans[1:0]	O	00	传输类型表示信号： 00：IDLE； 01：BUSY； 10：NONSEQ； 11：SEQ。 S802 仅支持 IDLE 和 NONSEQ 传输类型。
iahbl_pad_hwdata[31:0]	O	-	写数据总线： 32 位写数据总线。
iahbl_pad_hwrite	O	0	读写表示信号： 1：指示是写总线传输； 0：指示是读总线传输。
pad_iahbl_hrdata[31:0]	I	-	读数据总线： 32 位读数据总线。
pad_iahbl_hready	I	-	传输完成指示信号： 有效时指示当前传输已完成，CPU 将总线置于待命状态。
pad_iahbl_hresp	I	-	传输应答信号： 0：OKAY； 1：ERROR。
pad_bmu_iahbl_base[11:0]	I	-	IAHB-Lite 基址控制信号，上电复位之后需固定
pad_ bmu_iahbl_mask[11:0]	I	-	IAHB-Lite 地址对齐控制信号，上电复位之后需固定

9.4. 指令与数据的访问顺序

一旦 CPU 配置了高速缓存，由于 Cache 只映射指令总线，所有访问系统总线的数据

都会绕过 Cache 直接从总线矩阵访问系统总线。针对访问指令总线的数据，所有不可高缓的数据都会绕过 Cache 直接从总线矩阵访问指令总线，如图中的通道 1 所示；可高缓的数据会首先访问 Cache，如果命中则直接从 Cache 中取回数据，如果不命中，则会由 Cache 向指令总线接口发起请求，具体如图中的通道 2 所示。



图表 9-9 访问外总线顺序

10. 调试接口

10.1. 概述

调试接口是软件与处理器交互的通道。用户可以通过调试接口获取 CPU 的寄存器以及存储器内容等信息，包括其他的片上设备信息。此外，程序下载等操作也可以通过调试接口完成的。

为了满足低成本的应用需求，节省 CPU 外部引脚，CSKY 定义了一套调试接口，JTAG2 接口。JTAG2 调试接口包括 JTAG2 通信协议，JTAG2 接口控制器。S802 支持 2 线制 JTAG 协议，调试接口使用 JTAG2 协议与外部的调试器通信。

调试接口的主要特性如下：

- 支持 2 线制 JTAG 接口；
- 非侵入式获取 CPU 状态；
- 支持同步调试和异步调试，保证在极端恶劣情况下使处理器进入调试模式；
- 支持软断点；
- 可以设置多个内存断点；
- 检查和设置 CPU 寄存器的值；
- 检查和改变内存值；
- 可进行指令单步执行或多步执行；
- 快速下载程序；
- 可在 CPU 复位之后或在普通用户模式下进入调试模式；
- 可使用 TCIP 接口访问调试寄存器资源，详见调试手册；
- 支持 JTAG 或 TCIP 接口直接操作 HAD 寄存器发起内存访问请求，详见紧耦合 IP 手册；

C-SKY CPU 的调试工作是调试软件，调试代理服务程序，调试器和调试接口一起配合完成的，调试接口在整个 CPU 调试环境中的位置如图 9-1 所示。其中，调试软件和调试代理服务程序通过网络互联，调试代理服务程序与调试器通过 USB 连接，调试器与 CPU 的调试接口以 JTAG 模式通信。

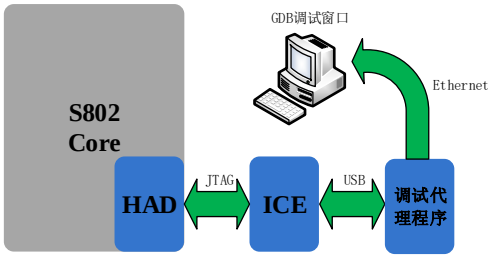


图 10-1 调试接口在整个 CPU 调试环境中的位置

10.2. 外部接口

调试模块与外部的接口主要是与 JTAG 相关的接口信号和调试相关的接口信号。图表 10-1 列出了与调试相关的接口信号。

图表 10-1 调试模块与外部的接口信号

信号名	方向
(sysio_pad_dbg_b)	输出
pad_had_jdb_req_b	输入
had_pad_jdb_ack_b	输出
pad_had_jtg_tap_en	输入
had_pad_jtg_tap_on	输出
had_pad_jdb_pm[1:0]	输出
pad_had_jtg_tclk	输入
pad_had_jtg_trst_b	输入
pad_had_jtg_tms_i	输入
had_pad_jtg_tms_o	输出
had_pad_htg_tms_oe	输出

1. biu_pad_dbg_b (sysio_pad_dbg_b)

低电平表示 CPU 处于调试模式中。

2. pad_had_jdb_req_b 与 had_pad_jdb_ack_b

pad_had_jdb_req_b 信号是让 CPU 异步进入调试状态的请求信号，该信号至少要维持低电平两个 JTAG 时钟周期才能保证 CPU 进入调试状态并且能够调试程序。如果该信号维持低电平不足两个 JTAG 时钟周期，那么可能会出现 CPU 已进入调试状态但不能调试程序的情况，因为该信号还会用于使能调试接口中的 TAP 状态机。

在 CPU 进入调试状态之后，had_pad_jdb_ack_b 信号会维持两个 JTAG 时钟周期的低电平以作为应答。

3. pad_had_jtg_tap_en 与 had_pad_jtg_tap_on

pad_had_jtg_tap_en 信号为调试接口中 TAP 状态机的使能信号，维持该信号为高电平至少一个 JTAG 时钟周期可以使能调试接口中的 TAP 状态机。如果该信号在 CPU 上电之后一直无效，那么使用同步方式（如设置调试接口寄存器 HCR 中的 DR 位，断点等）使 CPU 进入调试状态时可能无法调试程序。

在 TAP 状态机启动之后，had_pad_jtg_tap_on 信号将拉高。

4. had_pad_jdb_pm[1:0]

had_pad_jdb_pm[1:0]信号指示 CPU 当前低功耗模式，可以通过该信号确定 CPU 是否已进入调试模式。具体如图表 10-2 所示。

图表 10-2 had_pad_jdb_pm 指示当前 CPU 状态

had_pad_jdb_pm[1:0]	说明
00	普通模式
01	低功耗模式（STOP，DOZE，WAIT）
10	调试模式
11	保留

5. pad_had_jtg_tclk

JTAG 时钟信号。该信号为外部输入信号，一般为调试器产生的时钟信号，要保证该时钟信号的频率低于 CPU 时钟信号的频率 1/2 才能保证调试模块与 CPU 核之间的正常工作。

6. pad_had_jtg_trst_b

pad_had_jtg_trst_b 信号为 JTAG 复位信号，可以复位 TAP 状态机以及其他相关控制信号。

7. JTAG_2 相关信号

pad_had_jtg_tms_i 信号为 2 线制 JTAG 串行数据输入信号，调试接口在 JTAG 时钟信号 TCLK 的上升沿对其采样，而外部调试器在 JTAG 时钟的下降沿设置该信号；

该信号在空闲时必须保持为高电平，同时空闲时时钟信号最好停止。用户可以利用该信号同步复位 HAD 逻辑：在实现 2 线制 JTAG 接口的调试模块中，如果时钟信号一直有效，用户只需保持该信号为高电平状态并维持 80 个时钟周期即可同步复位调试模块。复位调试模块后，调试模块的 TAP 状态机回到 RESET 态，同时调试模块寄存器 HACR 复位到 0x82（指向 ID 寄存器）。

had_pad_jtg_tms_o 信号为 2 线制 JTAG 串行数据输出信号，调试接口在 JTAG 时钟信号 TCLK 的下降沿对其设置，而外部调试器在 JTAG 时钟的上升沿对其采样；

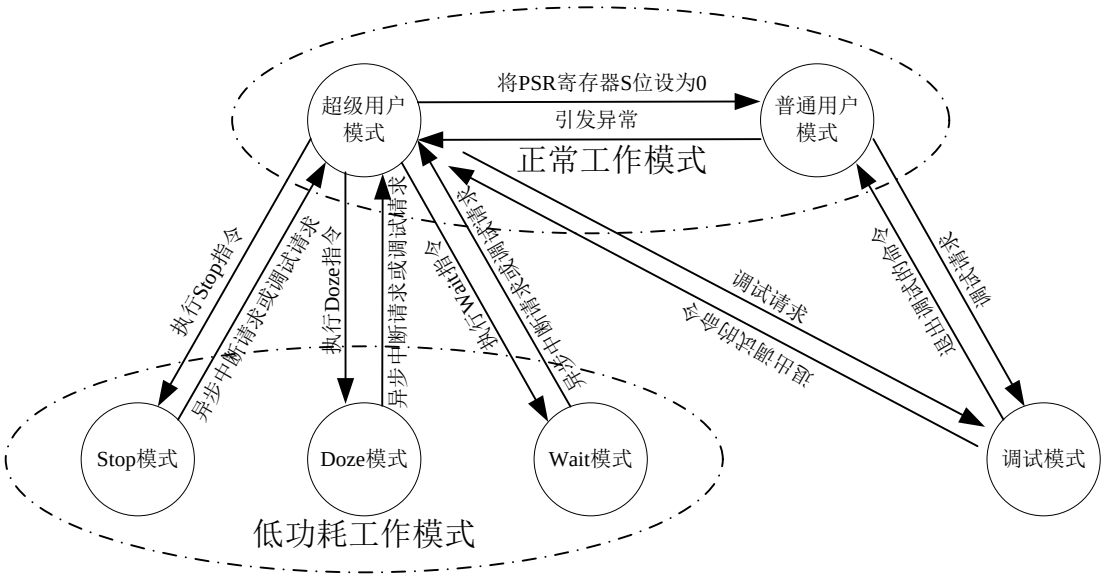
had_pad_jtg_tms_oe 信号为 had_pad_jtg_tms_o 信号有效指示信号。CPU 外部应利用该信号将 pad_had_jtg_tms_i 和 had_pad_jtg_tms_o 信号合为一个双向端口信号。

11. 工作模式转换

S802 总共有三类工作模式：正常运行模式、低功耗工作模式和调试模式，其中低功耗工作模式分为三种：STOP 模式、DOZE 模式、WAIT 模式，这三种低功耗模式的不同是由 SOC 设计者定义的。本章将详细解析 CPU 的工作模式以及模式之间的转换。

11.1. S802 工作模式及其转换

如图表 10-1 所示，S802 的工作模式有三类，即正常工作模式、低功耗工作模式和调试模式，CPU 处于哪种工作模式可以通过查询 had_pad_jdb_pm[1:0]信号得到。



图表 11-1 CPU 的各种工作状态示意图

11.2. 正常工作模式

CPU 的正常工作模式可以分为两种，超级用户模式和普通用户模式，CPU 处于哪种正常工作模式通过查询 PSR 寄存器中的 S 位得到，当 S 位为 1 时，CPU 工作于超级用户模式；当 S 位为 0 时，CPU 工作于普通用户模式。当 CPU 工作在超级用户模式时，可以通过将 S 位设置为 0 进入普通用户模式；当 CPU 工作于普通用户模式时，通过引发异常进入超级用户模式。

11.3. 低功耗模式

当 CPU 执行完低功耗指令(STOP、DOZE、WAIT)之后，CPU 将进入低功耗模式。三条指令执行的过程是，CPU 执行到低功耗指令后将等待前面的所有指令执行完，然后完成低功耗指令，同时根据低功耗指令类型拉起 sysio_pad_lpmd_b[1:0]信号，停止执行指令并冻结流水线，关掉内部时钟。

只有异步中断请求（pad_sysio_intraw_b 和 pad_sysio_fintraw_b 信号）或者调试请求才能使 CPU 退出低功耗模式，然后 CPU 从进入低功耗模式的低功耗指令处继续执行后续

指令。当前的 CPU 工作于哪种低功耗模式可以通过查询信号线 sysio_pad_lpmd_b[1:0]来得到，每种模式具体对应的场景由 SOC 设计者决定。

11.4. 调试模式

11.4.1. 调试模式

CPU 进入调试模式后将停止取指和执行指令，处于一种等待状态。该状态下 CPU 只执行 HAD 输入的指令，通过 HAD 输入指令可以查询和修改 CPU 的状态，进而进行调试。

11.4.2. 进入调试模式

当 CPU 接到调试请求之后，进入调试模式，其中的调试请求源可以有以下 5 种：

- 当 S802 HCR 的 ADR 位有效时，处理器直接进入调试模式。
- 当 S802 HCR 的 DR 位有效时，处理器在完成当前指令后进入调试模式。
- 当 S802 CSR 的 FDB 位有效时，处理器在执行 bkpt 指令进入调试模式。
- 当 S802 HCR 的 TME 位有效时，处理器在跟踪计数器的值减到 0 后进入调试模式。
- 在 S802 存储器断点调试模式下，当 BKPTA 或 BKPTB 被触发（即 MBCA 或 MBCB 的值为 0 时），或者 BKPTC—BKPTI 中有一个被触发，如果当前执行的指令符合断点要求，处理器进入调试模式。

处理器执行完当前的指令，保存流水线的信息，然后进入调试模式。当处理器处于低功耗模式时，通过设置 HCR 中的 ADR 以及 DR 的调试请求，可以使得处理器退出低功耗并进入调试模式。进入调试模式后，CPU 停止执行当前指令，等待用户通过调试接口输入有效的指令用于执行或退出调试模式。

11.4.3. 退出调试模式

如果 S802 HACR 的 GO、EX 位被置为 1，同时 R/W 为 0（写操作），RS 选择的是 WBBR、PSR、PC、IR、CSR 或者 Bypass 寄存器，则执行指令时 CPU 退出调试模式，进入正常工作模式。

注意：由于在调试模式下 PC，CSR，PSR 是可变的，因此在退出调试模式时，上述寄存器中的值必须是刚进入调试模式之时保存过的值。

12. 初始化参考代码

12.1. MPU 设置示例

```
/*设置区域 0 属性，可执行，非安全，超级用户和普通用户可读写
mfcr    r10,cr<19,0>
bclri   r10,0       //设置控制寄存器第0位为0，区域0可执行
bseti   r10,8       //设置控制寄存器8和9位为1，区域0超级用户和普通用户都可读写
bseti   r10,9
bclri   r10,24      //设置控制寄存器第24位为0，区域0为非安全区域
mtcr    r10,cr<19,0>

/*设置区域 1 属性，不可执行，安全，超级用户和普通用户可读写
mfcr    r10,cr<19,0>
bseti   r10,1       //设置控制寄存器第1位为1，区域1不可执行
bseti   r10,10      //设置控制寄存器10和11位为1，区域1超级用户和普通用户都可读
写
bseti   r10,11
bseti   r10,25      //设置控制寄存器第25位为1，区域1为安全区域
mtcr    r10,cr<19,0>

/*设置区域2属性，不可执行，非安全，超级用户可读写，普通用户只读
mfcr    r10,cr<19,0>
bseti   r10,2       //设置控制寄存器第2位为1，区域2不可执行
bclri   r10,12      //设置12位为0，13位为1，超级用户可读写，普通用户只读
bseti   r10,13
bclri   r10,26      //设置控制寄存器第26位为0，区域2非安全区域
mtcr    r10,cr<19,0>

/*设置区域3属性，可执行，安全，超级用户和普通用户都可读写
mfcr    r10,cr<19,0>
bclri   r10,3       //设置控制寄存器第3位为0，区域3可执行
bseti   r10,14      //设置14，15位为1，区域3超级用户和普通用户都可读写
bclri   r10,15
bseti   r10,27      //设置控制寄存器第27位为1，区域3为安全区域
```



```
mtcr      r10,cr<19,0>

/*区域4~7属性设置和区域0、1、2、3的属性设置相同

//区域0~7的可执行属性，设置控制寄存器CR<19,0>的[7:0]位
//区域0~7的访问权限，设置控制寄存器CR<19,0>的[23:8]位
//区域0~7的安全属性，通过设置控制寄存器CR<19,0>的[31:24]位

/*  设置保护区域0，基地址和保护区大小
movi      r10,0
mtcr      r10,cr<21,0>          //选择保护区域0
movi      r10,0x0               //设置保护区基地址0x00000000
ori       r10,r10,0x3f          //设置保护区大小 4G
mtcr      r10,cr<20,0>          //设置保护区基地址为 0，保护区大小为 4G

/*  设置保护区域1，基地址和保护区大小
movi      r10,1
mtcr      r10,cr<21,0>          //选择保护区域1
movih     r10,0x2800            //设置保护区基地址0x28000000
ori       r10,r10,0x2f          //设置保护区大小 16M
mtcr      r10,cr<20,0>          //设置保护区地址区间为0x28000000 ~ 0x29000000

/*设置保护区域2，基地址和保护区大小
movi      r10,2
mtcr      r10,cr<21,0>          //选择保护区域2
movih     r10,0x2800            //设置保护区基地址0x28000000
ori       r10,r10,0x27          //设置保护区大小 1M
mtcr      r10,cr<20,0>          //设置保护区地址区间为0x28000000 ~0x28100000

/*设置保护区域3，基地址和保护区大小
movi      r10,3
mtcr      r10,cr<21,0>          //选择保护区域2
movih     r10,0x28f0            //设置保护区基地址0x28f00000
ori       r10,r10,0x27          //设置保护区大小为 1M
mtcr      r10,cr<20,0>          //设置保护区地址区间为0x28f00000 ~0x29000000
```



```
    //保护区域3~7的基地址和保护区大小设置和区域0、1、2的设置相同
    //设置控制寄存器CR<21,0>选择保护区域
    //将保护区基地址和保护区大小，写入控制寄存器CR<20,0>，

    /* 使能MPU
    mfcr    r7, cr<18,0>        //选择MUP使能控制寄存器
    bseti   r7, 0               //设置控制寄存器CR<18,0>最低两位为2'b01,使能MPU
    bclri   r7, 1
mtcr    r7, cr<18,0>        //将预置值写入MPU使能控制寄存器，开启MPU
```

12.2. 高速缓存设置示例

开启高速缓存之前需要对整个高速缓存无效化，然后再根据实际应用需求配置 CER 并使能 Cache。

```
// 高速缓存无效化
    lrw   r1， 1                  // 预设第 0 位 INV_ALL 为 1
    lrw   r2， 0xe000f004         // 预设 CIR 所在地址
    st.w  r1， (r2，0x0)          // 将预设值写入 CIR，无效化操作开始
// 设置可高速缓存区域 crcr0
    lrw   r1， 0x00000039         // 预设起始地址为 0x0 的 512M 地址空间可高速缓存,
    lrw   r2， 0xe000f008         // 预设 CRCR0 所在地址
    st.w  r1， (r2，0x0)          // 将预设值写入 CIR，无效化操作开始
//可缓冲区域 crcr1~3 设置同 crcr0 ，将可高速缓冲的起始地址和 size 及使能位分别写到
//     crcr1~crcr3 对应的地址
//     crcr1 地址 0XE000F00C，
//     crcr2 地址 0XE000F010
//     crcr3 地址 0XE000F014

// 开启高速缓存
    lrw    r1，0                  // 预设值为 0x0000000
    bseti  r1，0                  //设置 Cache enable 打开
    bclri  r1，1                  //设置，指令和数据都可高速缓冲
//CER 只有低两位可配置，其他位为保留位，设置无效
    lrw r2， 0xe000f000           // 预设 CER 所在地址
    st.w r1， (r2,0x0)            // 将预设值写入 CER，Cache 使能
```


12.3. 中断使能初始化

在配置好中断控制器和中断向量表之后（具体参考紧耦合 IP 手册），需要将中断使能位打开，具体设置如下：

```
mfcr    r1, cr<0,0>
bseti   r1, 6
bseti   r1, 8
mtcr    r1, cr<0,0>
```

12.4. 通用寄存器初始化示例

//初始化通用寄存器 r0~r15 和 r28 为 0。

```
movi    r0，0       //初始化通用寄存器 0 为 0
movi    r1，0
movi    r2，0
movi    r3，0
movi    r4，0
movi    r5，0
movi    r6，0
movi    r7，0
movi    r8，0
movi    r9，0
movi    r10，0
movi    r11，0
movi    r12，0
movi    r13，0
movi    r14，0
movi    r15，0
movi    r28，0
```

12.5. 堆栈指针初始化示例

堆栈指针的设置和程序当前处在状态有关系如下所示，超级用户态下堆栈指针初始化为示例 1，用户态下堆栈指针初始化为示例 2。

示例 1：

//超级用户态下，设置超级用户态和用户态堆栈指针

//超级用户态下 r14 映射为超级用户态的堆栈指针寄存器

//超级用户态下，用户态堆栈指针寄存器为 CR<14,1>

```
lrw     r14，0x01000000 //设置超级用户态指针
lrw     r0，  0x02000000 //
mtcr    r0，  cr<14,1>     //设置用户态堆栈指针
```

示例 2：

//用户态下，只能设置用户态指针：

//用户态下 r14 映射为用户态的堆栈指针寄存器

```
lrw    r14,   0x02000000 //设置用户态指针
```

12.6. 异常和中断服务程序入口地址设置示例

异常和中断服务程序的入口地址设置分两个步骤：

Step1： 设置异常向量表地址寄存器 VBR，根据向量号，各个异常向量号对应异常向量表地址为，VBR + （向量号<< 2 ）

Step2：将异常服务程序的入口地址写到 Step1 中异常向量号对应的异常向量表地址中。

异常向量号为 2 的访问错误异常示例如下：

//设置异常向量表入口地址，VBR 对应控制寄存器 cr<1,0>

```
Lrw     r2，  0
mtcr     r2,     cr<1,0>                    //设置异常向量表的地址为 0
```

//异常/中断服务程序入口地址设置

```
lrw     r1, ACCERR_ERROR_BEGIN     //设置异常服务程序入口地址
lrw     r2, 0                                  //VBR  地址
movi    r3, 0x2                               //异常向量号
lsli      r3，r3，2
addu    r2，r2，r3                          //计算异常向量号对应异常向量表地址
st.w     r1,(r2，0x0)                        //将异常服务程序入口地址，存入相应的异常向量表地址中
br     START
```



```
//用户设置的访问错误异常服务程序
label   ACCERR_ERROR_BEGIN
/*用户的异常服务程序*/
label   ACCERR_ERROR_END
label   START
```

附录 A 指令术语表

以下是每条 S802 实现的 CSKY V2 指令的具体描述，下面根据指令英文字母顺序对每条指令进行详细说明。

每条指令助记符结尾以数字“32”或“16”表示指令位宽。例如，“addc32”表示该指令为 32 位无符号带进位加法指令，“addc16”表示该指令为 16 位无符号带进位加法指令。

如果省略助记符中的指令位宽（如“addc”），系统会自动将汇编为最优化的指令。

其中，指令中文名称中带#的为伪指令。

ADDC——无符号带进位加法指令

统一化指令

语法	操作	编译结果

addc rz, rx	RZ ← RZ + RX + C，
	C←进位
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		addc16 rz, rx;
		else
		addc32 rz, rz, rx;

addc rz, rx, ry	RZ ← RX + RY + C，
	C←进位
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		addc16 rz, rx;
		else
		addc32 rz, rx, ry;

说明： 将 RZ/RY、RX 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。

影响标志位： C ← 进位

异常： 无

16位指令

操作： RZ ← RZ + RX + C，C←进位

语法： addc16 rz, rx

说明： 将 RZ、RX 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。

影响标志位： C ← 进位

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 0	RZ	RX	0 1

32位指令

操作： RZ ← RX + RY + C，C←进位

语法： addc32 rz, rx, ry

说明： 将 RX、RY 与 C 位的值相加，并把结果存在 RZ，进位存在 C 位。

影响标志位： C ← 进位

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 0 0 0 0		0 0 0 1 0		RZ	

ADDI——无符号立即数加法指令

统一化指令

语法	操作	编译结果

addi rz, oimm12	RZ ← RZ + zero_extend(OIMM12)	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8) and (oimm12<257),
		addi16 rz, oimm8;
		else
		addi32 rz, rz, oimm12;

addi rz, rx, oimm12	RZ ← RX + zero_extend(OIMM12)	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (oimm12<9) and (z<8) and (x<8),
		addi16 rz, rx, oimm3;
		elsif (oimm12<257) and (x==z) and (z<8),
		addi16 rz, oimm8;
		else
		addi32 rz, rx, oimm12;

说明： 将带偏置 1 的立即数零扩展至 32 位，然后与 RX/RZ 的值相加，把结果存入 RZ。

影响标志位： 无影响

限制： 若源寄存器是 R28，立即数的范围为 0x1-0x40000。

若源寄存器不是 R28，立即数的范围为 0x1-0x1000。

异常： 无

16位指令----1

操作： RZ ← RZ + zero_extend(OIMM8)

语法： addi16 rz, oimm8

说明： 将带偏置 1 的 8 位立即数（OIMM8）零扩展至 32 位，然后与 RZ 的值相加，把结果存入 RZ。

注意：二进制操作数 IMM8 等于 OIMM8 - 1。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 1-256。

异常： 无

指令格式：

15 14 11 10 8 7 0

0	0 1 0 0	RZ	IMM8

IMM8 域——指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM8 比起二进制操作数 IMM8 需偏置 1。

00000000——加 1

00000001——加 2

……

11111111——加 256

16位指令----2

操作： RZ ← RX + zero_extend(OIMM3)

语法： addi16 rz, rx, oimm3

说明： 将带偏置 1 的 3 位立即数（OIMM3）零扩展至 32 位，然后与 RX 的值相加，把结果存入 RZ。

注意：二进制操作数 IMM3 等于 OIMM3 - 1。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 1-8。

异常： 无

指令格式：

15 14 10 8 7 5 4 2 1 0

0	1 0 1 1	RX	RZ	IMM3	1 0

IMM3 域——指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM3 比起二进制操作数 IMM3 需偏置 1。

000——加 1

001——加 2

……

111——加 8

32位指令

操作： RZ ← RX + zero_extend(OIMM12)

语法： addi32 rz, rx, oimm12

说明： 将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后与 RX 的值相加，把结果存入 RZ。

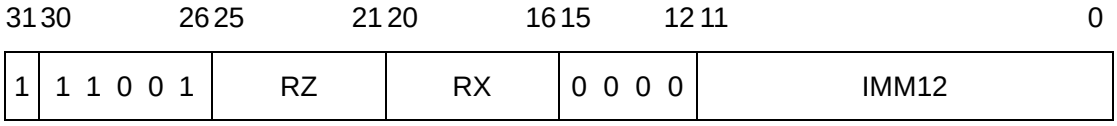
注意：二进制操作数 IMM12 等于 OIMM12 - 1。

影响标志位： 无影响

限制： 立即数的范围为 0x1-0x1000。

异常： 无

指令格式：



IMM12 域——指定不带偏置立即数的值。

注意：加到寄存器里的值 OIMM12 比起二进制操作数 IMM12 需偏置 1。

000000000000——加 0x1

000000000001——加 0x2

……

111111111111——加 0x1000

ADDI(SP)——无符号（堆栈指针）立即数加法指令

统一化指令

语法	操作	编译结果
addi rz, sp, imm	RZ ← SP + zero_extend(IMM)	仅存在 16 位指令。 addi rz, sp, imm
addi sp, sp, imm	SP ← SP+ zero_extend(IMM)	仅存在 16 位指令。 addi sp, sp, imm

说明： 将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ 或者 SP。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 0x0-0x3fc。

异常： 无

16位指令----1

操作： RZ ← SP + zero_extend(IMM)

语法： addi16 rz, sp, imm8

说明： 将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ。

注意：立即数（IMM）等于二进制操作数 IMM8 << 2。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为(0x0-0xff) << 2。

异常： 无

指令格式：

15 14 11 10 8 7 0

0	0 0 1 1	RZ	IMM8

IMM8 域——指定不带移位立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数 IMM8 需左移 2 位。

00000000——加 0x0

00000001——加 0x4

……

11111111——加 0x3fc

16位指令----2

操作： SP ← SP + zero_extend(IMM)

语法： addi16 sp, sp, imm

说明： 将立即数（IMM）零扩展至 32 位，然后与堆栈指针（SP）的值相加，把结果存入 RZ。

注意：立即数（IMM）等于二进制操作数{IMM2, IMM5} << 2。

影响标志位： 无影响

限制： 源与目的寄存器均为堆栈指针寄存器（R14）；立即数的范围为(0x0-0x7f) << 2。

异常： 无

指令格式：

15	14 11	10	9 8	7 5	4 0
0	0 0 1 0	1	IMM2	0 0 0	IMM5

IMM 域——指定不带移位立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数{IMM2, IMM5}需左移 2 位。

{00, 00000}——加 0x0

{00, 00001}——加 0x4

……

{11, 11111}——加 0x1fc

ADDU——无符号加法指令

统一化指令

语法	操作	编译结果

addu rz, rx	RZ ← RZ + RX	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<16) and (x<16), then
		addu16 rz, rx;
		else
		addu32 rz, rz, rx;

addu rz, rx, ry	RZ ← RX + RY	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8) and (x<8) and (y<8), then
		addu16 rz, rx, ry;
		elsif (y==z) and (x<16) and (z<16), then
		addu16 rz, rx;
		else
		addu32 rz, rx, ry;

说明： 将 RZ/RY 与 RX 的值相加，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

16位指令----1

操作： RZ ← RZ + RX

语法： addu16 rz, rx

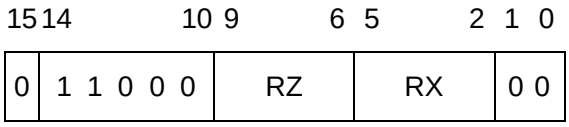
说明： 将 RZ 与 RX 的值相加，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：



16位指令----2

操作： RZ ← RX + RY

语法： addu16 rz, rx, ry

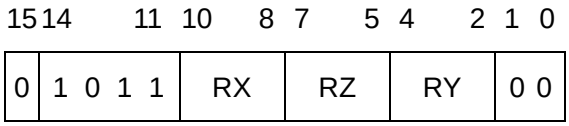
说明： 将 RX 与 RY 的值相加，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 无

指令格式：



32 位指令

操作： RZ ← RX + RY

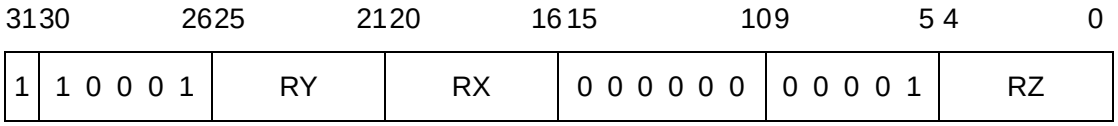
语法： addu32 rz, rx, ry

说明： 将 RX 与 RY 的值相加，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

指令格式：



AND——按位与指令

统一化指令

语法	操作	编译结果

and rz, rx	RZ ← RZ and RX	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		and16 rz, rx;
		else
		and32 rz, rz, rx;

and rz, rx, ry	RZ ← RX and RY	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		and16 rz, rx;
		else
		and32 rz, rx, ry;

说明： 将 RZ/RY 与 RX 的值按位与，并把结果存在 RZ；

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ and RX

语法： and16 rz, rx

说明： 将 RZ 与 RX 的值按位与，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 0	RZ	RX	0 0

32位指令

操作： RZ ← RX and RY

语法： and32 rz, rx, ry

说明： 将 RX 与 RY 的值按位与，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10 9	8 7 6 5 4	3 2 1 0
1	1 0 0 0 1	RY	RX	0 0 1 0 0 0	0 0 0 0 1	RZ

ANDI——立即数按位与指令

统一化指令

语法	操作	编译结果
andi rz, rx, imm12	RZ ← RX and zero_extend(IMM12)	仅存在 32 位指令 andi32 rz, rx, imm12

说明： 将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位与操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFF。

异常： 无

32位指令

操作： RZ ← RX and zero_extend(IMM12)

语法： andi32 rz, rx, imm12

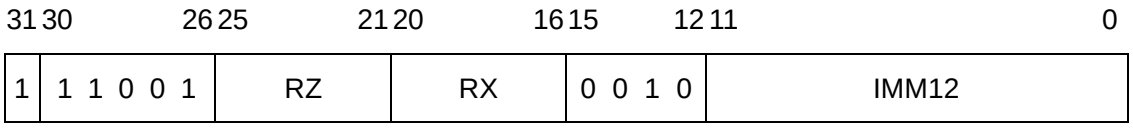
说明： 将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位与操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFF。

异常： 无

指令格式：



ANDN——按位非与指令

统一化指令

语法	操作	编译结果

andn rz, rx	RZ ← RZ and (!RX)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		andn16 rz, rx;
		else
		andn32 rz, rz, rx;

andn rz, rx, ry	RZ ← RX and (!RY)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		andn16 rz, ry;
		else
		andn32 rz, rz, rx;

说明： 对于 andn rz, rx，将 RZ 的值与 RX 的非值按位与，并把结果存在 RZ；对于 andn rz, rx, ry，将 RX 的值与 RY 的非值按位与，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ and (!RX)

语法： andn16 rz, rx

说明： 将 RZ 的值与 RX 的非值按位与，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 1 0	RZ	RX	0 1

32位指令

操作： RZ ← RX and (!RY)

语法： andn32 rz, rx, ry

说明： 将 RX 的值与 RY 的非值按位与，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 1 0 0 0		0 0 0 1 0		RZ	

ANDNI——立即数按位非与指令

统一化指令

语法	操作	编译结果
andni rz, rx, imm12	RZ ← RX and !(zero_extend(IMM12))	仅存在 32 位指令 andni32 rz, rx, imm12

说明： 将 12 位立即数零扩展至 32 位并取非，然后与 RX 的值进行按位与操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFF。

异常： 无

32位指令

操作： RZ ← RX and !(zero_extend(IMM12))

语法： andni32 rz, rx, imm12

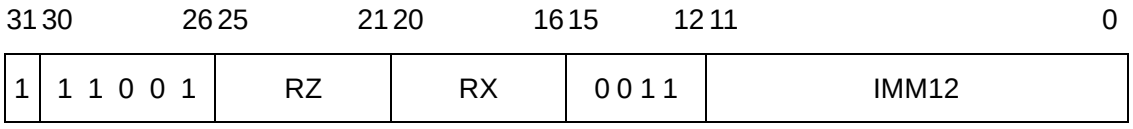
说明： 将 12 位立即数零扩展至 32 位并取非，然后与 RX 的值进行按位与操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFF。

异常： 无

指令格式：



ASR——算术右移指令

统一化指令

语法	操作	编译结果

asr rz, rx	RZ ← RZ >>> RX[5:0]	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		asr16 rz, rx;
		else
		asr32 rz, rz, rx;

asr rz, rx, ry	RZ ← RX >>> RY[5:0]	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		asr16 rz, ry;
		else
		asr32 rz, rx, ry;

说明： 对于 asr rz, rx，将 RZ 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0]的值大于 30，那么 RZ 的值（0 或-1）由 RZ 原值的符号位决定；

对于 asr rz, rx, ry，将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0]的值大于 30，那么 RZ 的值（0 或-1）由 RX 的符号位决定。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ >>> RX[5:0]

语法： asr16 rz, rx

说明： 将 RZ 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0]的值大于 30，那么 RZ 的值（0 或-1）由 RZ 原值的符号位决定。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 0	RZ	RX	1 0

32位指令

操作： RZ ← RX >>> RY[5:0]

语法： asr32 rz, rx, ry

说明： 将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入RZ，右移位数由RY低6位（RY[5:0]）的值决定；如果RY[5:0]的值大于 30，那么 RZ 的值（0 或-1）由 RX 的符号位决定。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 1 0 0	RZ

ASRC——立即数算术右移至 C 位指令

统一化指令

语法	操作	编译结果
asrc rz, rx, oimm5	RZ ← RX >>> OIMM5, C ← RX[OIMM5 - 1]	仅存在 32 位指令 asrc32 rz, rx, oimm5

说明： 将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的符号位（最高位），RZ 的值（0 或-1）由 RX 的符号位决定。

影响标志位： C ← RX[OIMM5 - 1]

限制： 立即数的范围为 1-32。

异常： 无

32位指令

操作： RZ ← RX >>> OIMM5, C ← RX[OIMM5 - 1]

语法： asrc32 rz, rx, oimm5

说明： 将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的符号位（最高位），RZ 的值（0 或-1）由 RX 的符号位决定。注意：二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： C ← RX[OIMM5 - 1]

限制： 立即数的范围为 1-32。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 1		0 0 1 0 0		RZ	

IMM5 域——指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000——移 1 位

00001——移 2 位

……

11111——移 32 位

ASRI——立即数算术右移指令

统一化指令

语法	操作	编译结果

asri rz, rx, imm5	RZ ← RX >>> IMM5	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<8) and (z<8), then
		asri16 rz, rx, imm5;
		else
		asri32 rz, rx, imm5;

说明： 对 asri rz, rx, imm5 而言，将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RX >>> IMM5

语法： asri16 rz, rx, imm5

说明： 将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 0-31。

异常： 无

指令格式：

15	14 11	10 8	7 5	4 0
0	1 0 1 0	RX	RZ	IMM5

32位指令

操作： RZ ← RX >>> IMM5

语法： asri32 rz, rx, imm5

说明： 将 RX 的值进行算术右移（原值右移，左侧移入原符号位的拷贝），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 0		0 0 1 0 0		RZ	

BCLRI——立即数位清零指令

统一化指令

语法	操作	编译结果

bclri rz, imm5	RZ ← RZ[IMM5]清零	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8)，then
		bclri16 rz, imm5;
		else
		bclri32 rz, rz, imm5;

bclri rz, rx, imm5	RZ ← RX[IMM5]清零	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (z<8), then
		bclri16 rz, imm5;
		else
		bclri32 rz, rx, imm5;

说明： 将 RZ/RX 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ；

影响标志位： 无影响

限制： 立即数的范围为 0-31。

16位指令

操作： RZ ← RZ[IMM5]清零

语法： bclri16 rz, imm5

说明： 将 RZ 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；

立即数的范围为 0-31。

异常： 无

指令格式：

15 14 10 8 7 5 4 0

0	0 1 1 1	RZ	1 0 0	IMM5

32位指令

操作： RZ ← RX[IMM5]清零

语法： bclri32 rz, rx, imm5

说明： 将 RX 的值中，由 IMM5 域值所指示的位清零，其余位保持不变，把清零后的结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 0 1 0 1 0	0 0 0 0 1	RZ

BF——C 为 0 分支指令

统一化指令

语法	操作	编译结果

bf label	C 等于零则程序转移。
	if(C==0)
	PC←PC + sign_extend(offset << 1);
	else
	PC ← next PC;
		根据跳转的范围编译为对应的 16 位或 32 位指令。
		if (offset<1KB), then
		bf16 label;
		else
		bf32 label;

说明： 如果条件标志位 C 等于零，则程序转移到 label 处执行；否则程序执行下一条指令。

Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。BF 指令的转移范围是±64KB 地址空间。

影响标志位： 无影响

异常： 无

16位指令

操作： C 等于零则程序转移。

if(C==0)

PC ← PC + sign_extend(offset << 1)

else

PC ← PC + 2

语法： bf16 label

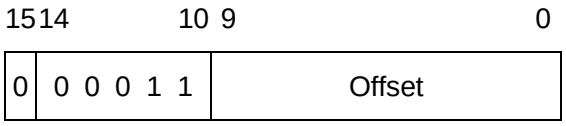
说明： 如果条件标志位 C 等于 0，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 2。

Label 由当前程序 PC 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BF16 指令的转移范围是±1KB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：



32位指令

操作： C 等于零则程序转移

if(C == 0)

PC ← PC + sign_extend(offset << 1)

else

PC← PC + 4

语法： bf32 label

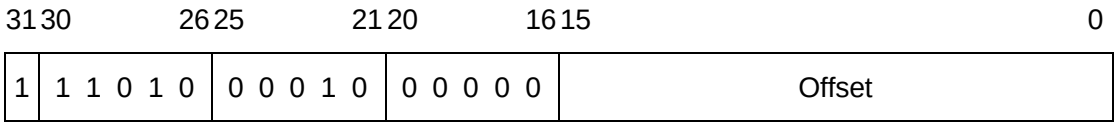
说明： 如果条件标志位 C 等于零，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。

Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BF 指令的转移范围是±64KB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：



BGENI——立即数位产生指令#

统一化指令

语法	操作	编译结果
bgeni rz, imm5	RZ ←（2）^{IMM5}	仅存在 32 位指令。 bgeni32 rz, imm5

说明： 对由 5 位立即数确定的 RZ 的位（RZ[IMM5]）置位，并清除 RZ 的其它位。

注意，如果 IMM5 小于 16，该指令是 movi rz, (2)^{IMM5} 的伪指令；

如果 IMM5 大于等于 16，该指令是 movih rz, (2)^{IMM5} 的伪指令。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作： RZ ←（2）^{IMM5};

语法： bgeni32 rz, imm5

说明： 对由 5 位立即数确定的 RZ 的位（RZ[IMM5]）置位，并清除 RZ 的其它位。

注意，如果 IMM5 小于 16，该指令是 movi32 rz, (2)^{IMM5} 的伪指令；

如果 IMM5 大于等于 16，该指令是 movih32 rz, (2)^{IMM5} 的伪指令。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

如果 IMM5 小于 16：

31	30 25	24 20	19 15	14 0
1	1 1 0 1 0	1 0 0 0 0	RZ	（2）^{IMM5}

如果 IMM5 大于等于 16：

31	30 25	24 20	19 15	14 0

1	1 1 0 1 0	1 0 0 0 1	RZ	（2）IMM5

BKPT——断点指令

统一化指令

语法	操作	编译结果
bkpt	引起一个断点异常或者进入调试模式	总是编译为 16 位指令。 bkpt16

说明： 断点指令

影响标志位： 无影响

异常： 断点异常

16位指令

操作： 引起一个断点异常或者进入调试模式

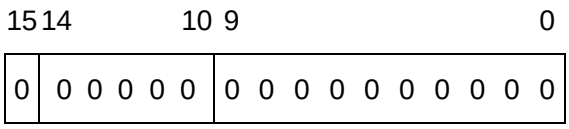
语法： bkpt16

说明： 断点指令

影响标志位： 无影响

异常： 断点异常

指令格式：



BMASKI——立即数位屏蔽产生指令

统一化指令

语法	操作	编译结果
bmaski rz, oimm5	RZ ← $(2)^{OIMM5}$ - 1	仅存在 32 位指令 bmaski32 rz, oimm5

说明： 产生连续低位为 1、高位为 0 的立即数，并将该立即数存入 RZ。带偏置的立即数 OIMM5 指定被置 1 的连续低位（RX[OIMM5-1:0]）的位数，其余高位清零。当 OIMM5 为 0 或 32 时，RX 所有位均被置 1。

注意，OIMM5 为 1-16 时由 movi 指令执行。

影响标志位： 无影响

限制： 立即数的范围为 0，17-32；

异常： 无

32位指令

操作： RZ ← $(2)^{OIMM5}$ - 1

语法： bmaski32 rz, oimm5

说明： 产生连续低位为 1、高位为 0 的立即数，并将该立即数存入 RZ。带偏置的立即数 OIMM5 指定被置 1 的连续低位（RX[OIMM5-1:0]）的位数，其余高位清零。当 OIMM5 为 0 或 32 时，RX 所有位均被置 1。

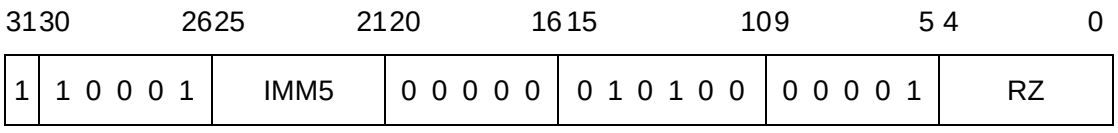
注意，OIMM5 为 1-16 时由 movi 指令执行；二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： 无影响

限制： 立即数的范围为 0，17-32；

异常： 无

指令格式：



IMM5 域——指定被置 1 的连续低位的最高位。

注意：立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

10000——0-16 位置位

10001——0-17 位置位

……

11111——0-31 位置位

BMCLR——BCTM 位清零指令

统一化指令

语法	操作	编译结果
bmclr	清除状态寄存器的 BM 位。 PSR(BM) ← 0	仅存在 32 位指令。 bmclr32

说明： PSR 的 BM 位被清零。

影响标志位： 无影响

异常： 无

注意： 该指令仅实现于支持二进制代码转译机制的S802处理器。

32位指令

操作： 清除状态寄存器的 BM 位

PSR(BM) ← 0

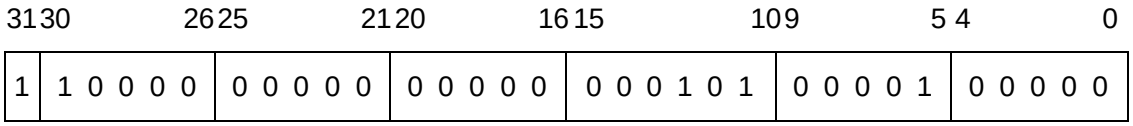
语法： bmclr32

说明： PSR 的 BM 位被清零。

影响标志位： 无影响

异常： 无

指令格式：



BPOP.H——二进制转译半字压栈指令

统一化指令

语法	操作	编译结果

bpop.h rz	更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载半字到寄存器 RZ 中；
	if (BSP - 2 < FP')
	R15 ← next PC
	PC ← SVBR - 12
	else
	BSP ← BSP - 2;
	RZ ← zero_extend(MEM[BSP]);
		仅存在 16 位指令
		bpop.h rz;

说明： 将二进制转译堆栈指针寄存器（BSP）减 2 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 2 的值小于 FP'，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的半字经过零扩展到 32 位后，加载到寄存器 RZ 中。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载半字到寄存器 RZ 中；

```
if (BSP - 2 < FP')
    R15 ← next PC
    PC ← SVBR - 12
else
   BSP ← BSP - 2;
   RZ ← zero_extend(MEM[BSP]);
```


语法： bpop16.h rz

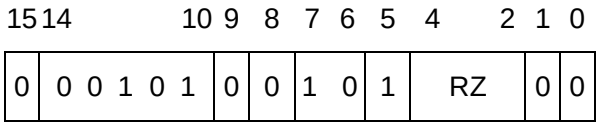
说明： 将二进制转译堆栈指针寄存器（BSP）减 2 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 2 的值小于 FP'，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的半字经过零扩展到 32 位后，加载到寄存器 RZ 中。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：



BPOP.W——二进制转译字压栈指令

统一化指令

语法	操作	编译结果

bpop.w rz	更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载字到寄存器 RZ 中；
	if (BSP - 4 < FP')
	R15 ← next PC
	PC ← SVBR - 12
	else
	BSP ← BSP - 4;
	RZ ← MEM[BSP];
		仅存在 16 位指令
		bpop.w rz;

说明： 将二进制转译堆栈指针寄存器（BSP）减 4 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 4 的值小于 FP'，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的字加载到寄存器 RZ 中。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端，然后从二进制转译堆栈存储器加载字到寄存器 RZ 中；

```
if (BSP - 4 < FP')
    R15 ← next PC
    PC ← SVBR - 12
else
   BSP ← BSP - 4;
   RZ ← MEM[BSP];
```

语法： bpop16.w rz

说明： 将二进制转译堆栈指针寄存器（BSP）减 4 的值与二进制转译帧指针寄存器（FP'）进行比较。如果 BSP 减 4 的值小于 FP'，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，更新 BSP 到二进制转译堆栈存储器的顶端，然后将二进制转译堆栈存储器中的字加载到寄存器 RZ 中。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

15	14 10	9	8	7	6	5	4 2	1	0
0	0 0 1 0 1	0	0	1	0	1	RZ	1	0

BPUSH.H——二进制转译半字压栈指令

统一化指令

语法	操作	编译结果

bpush.h rz	将寄存器 RZ 中的半字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；
	if (BSP + 2 > TOP)
	R15 ← next PC
	PC ← SVBR - 12
	else
	MEM[BSP] ← RZ[15:0];
	BSP ← BSP + 2;
		仅存在 16 位指令
		bpush.h rz;

说明： 将二进制转译堆栈指针寄存器（BSP）加 2 的值与二进制转译栈顶寄存器（TOP）进行比较。如果 BSP 加 2 的值大于 TOP，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，将寄存器 RZ 中的低半字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 将寄存器 RZ 中的半字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；

```
if (BSP + 2 > TOP)
    R15 ← next PC
    PC ← SVBR - 12
else
   MEM[BSP] ← RZ[15:0];
   BSP ← BSP + 2;
```

语法： bpush16.h rz

说明： 将二进制转译堆栈指针寄存器（BSP）加 2 的值与二进制转译栈顶寄存器（TOP）进行比较。如果 BSP 加 2 的值大于 TOP，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，将寄存器 RZ 中的低半字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

15	14 10	9	8	7 6	5	4 2	1 0
0	0 0 1 0 1	0	0	1 1	1	RZ	0 0

BPUSH.W——二进制转译字压栈指令

统一化指令

语法	操作	编译结果

bpush.w rz	将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；
	if (BSP + 4 > TOP)
	R15 ← next PC
	PC ← SVBR - 12
	else
	MEM[BSP] ← RZ[31:0];
	BSP ← BSP + 4;
		仅存在 16 位指令
		bpush.w rz;

说明： 将二进制转译堆栈指针寄存器（BSP）加 4 的值与二进制转译栈顶寄存器（TOP）进行比较。如果 BSP 加 4 的值大于 TOP，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新二进制转译堆栈寄存器到二进制转译堆栈存储器的顶端；

```
if (BSP + 4 > TOP)
    R15 ← next PC
    PC ← SVBR - 12
else
   MEM[BSP] ← RZ[31:0];
   BSP ← BSP + 4;
```

语法： bpush16.w rz

说明： 将二进制转译堆栈指针寄存器（BSP）加 4 的值与二进制转译栈顶寄存器（TOP）进行比较。如果 BSP 加 4 的值大于 TOP，则将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 SVBR-12 处执行；否则，将寄存器 RZ 中的字存储到二进制转译堆栈存储器中，然后更新 BSP 到二进制转译堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r0 – r7。

异常： 未对齐访问异常、未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	1	1	1	RZ			1	0

BMSET——BCTM 位置位指令

统一化指令

语法	操作	编译结果
bmset	设置状态寄存器的 BM 位。 PSR(BM) ← 1	仅存在 32 位指令。 bmset32

说明： PSR 的 BM 位被置位。

影响标志位： 无影响

异常： 无

注意： 该指令仅实现于支持二进制代码转译机制的S802处理器。

32位指令

操作： 设置状态寄存器的 BM 位

PSR(BM) ← 1

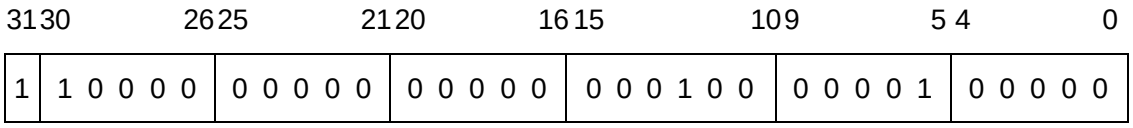
语法： bmset32

说明： PSR 的 BM 位被置位。

影响标志位： 无影响

异常： 无

指令格式：



BR——无条件跳转指令

统一化指令

语法	操作	编译结果

br label	PC ← PC + sign_extend(offset << 1)	根据跳转的范围编译为对应的 16 位或 32 位指令
		if(offset<1KB), then
		br16 label;
		else
		br32 label;

说明： 程序无条件跳转到 label 处执行。

Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。

影响标志位： 无影响

异常： 无

16位指令

操作： PC ← PC + sign_extend(offset << 1)

语法： br16 label

说明： 程序无条件跳转到 label 处执行。

Label 由当前程序 PC 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BR16 指令的跳转范围是±1KB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：

15	14	10	9	0
0	0 0 0 0 1		Offset	

32位指令

操作： PC ← PC + sign_extend(offset << 1)

语法： br32 label

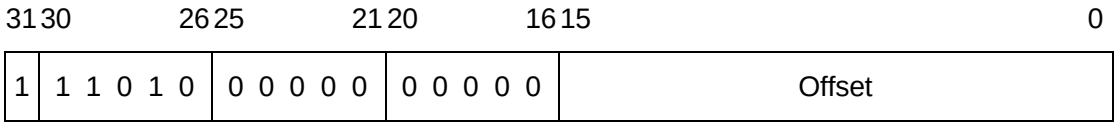
说明： 程序无条件跳转到 label 处执行。

Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BR 指令的跳转范围是±64KB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：



BSETI——立即数位置位指令

统一化指令

语法	操作	编译结果

bseti rz, imm5	RZ ← RZ[IMM5]置位	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8), then
		bseti16 rz, imm5；
		else
		bseti32 rz, rz, imm5；

bseti rz, rx, imm5	RZ ← RX[IMM5]置位	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if((x==z) and (z<8)，then
		bseti16 rz, imm5；
		else
		bseti32 rz, rx, imm5；

说明： 将 RZ/RX 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

16位指令

操作： RZ ← RZ[IMM5]置位

语法： bseti16 rz, imm5

说明： 将 RZ 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 0-31。

异常： 无

指令格式：

15 14 10 8 7 5 4 0

0	0 1 1 1	RZ	1 0 1	IMM5

32位指令

操作： RZ ← RX[IMM5]置位

语法： bseti32 rz, rx, imm5

说明： 将 RX 的值中，由 IMM5 域值所指示的位置 1，其余位保持不变，把置位后的结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 0 1 0 1 0		0 0 0 1 0		RZ	

BSR——跳转到子程序指令

统一化指令

语法	操作	编译结果
bsr label	链接并跳转到子程序： R15 ← next PC PC ← PC + sign_extend(offset << 1)	仅存在于 32 位指令 bsr32 label

说明： 子程序跳转，将子程序的返回地址（下一条指令的 PC）保存在链接寄存器 R15 中，程序转移到 label 处执行。

Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。

影响标志位： 无影响

异常： 无

32位指令

操作： 链接并跳转到子程序：

R15 ← PC+4

PC ← PC + sign_extend(offset << 1)

语法： bsr32 label

说明： 子程序跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序转移到 label 处执行。

Label 由当前程序 PC 加上左移 1 位的 26 位相对偏移量有符号扩展到 32 位后的值得到。BSR 指令的跳转范围是±64MB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：

31	30	26	25	0
1	1 1 0 0 0		Offset	

BT——C 为 1 分支指令

统一化指令

语法	操作	编译结果

bt label	if(C == 1)
	PC ← PC + sign_extend(offset << 1);
	else
	PC ← next PC;
		根据跳转的范围编译为对应的 16 位或 32 位指令。
		if (offset<1KB)，then
		bt16 label;
		else
		bt32 label;

说明： 如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令。

Label 由当前程序 PC 加上左移 1 位的相对偏移量有符号扩展到 32 位后的值得到。BT 指令的转移范围是±64KB 地址空间。

影响标志位： 无影响

异常： 无

16位指令

操作： C 等于一则程序转移

if(C == 1)

PC ← PC + sign_extend(offset << 1)

else

PC ← PC + 2

语法： bt16 label

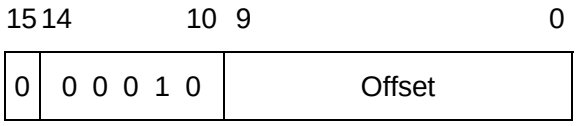
说明： 如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 2。

Label 由当前程序 PC 加上左移 1 位的 10 位相对偏移量有符号扩展到 32 位后的值得到。BT16 指令的转移范围是±1KB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：



32位指令

操作： C 等于一则程序转移

if(C == 1)

PC ← PC + sign_extend(offset << 1)

else

PC ← PC + 4

语法： bt32 label

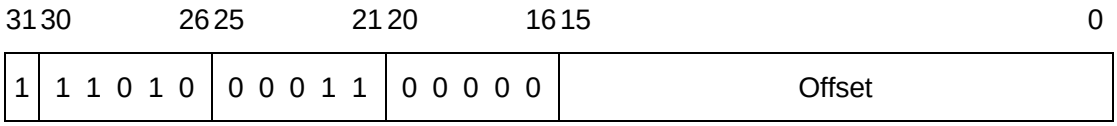
说明： 如果条件标志位 C 等于 1，则程序转移到 label 处执行；否则程序执行下一条指令，即 PC ← PC + 4。

Label 由当前程序 PC 加上左移 1 位的 16 位相对偏移量有符号扩展到 32 位后的值得到。BT 指令的转移范围是±64KB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：



BTSTI——立即数位测试指令

统一化指令

语法	操作	编译结果
btsti rx, imm5	C ← RX[IMM5]	仅存在 32 位指令 btsti32 rx, imm5

说明： 对由 IMM5 决定的 RX 的位（RX[IMM5]）进行测试，并使条件位 C 的值等于该位的值。

影响标志位： C ← RX[IMM5]

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作： C ← RX[IMM5]

语法： btsti32 rx, imm5

说明： 对由 IMM5 决定的 RX 的位（RX[IMM5]）进行测试，并使条件位 C 的值等于该位的值。

影响标志位： C ← RX[IMM5]

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 0 1 0 1 0		0 0 1 0 0		0 0 0 0 0	

CMPHS——无符号大于等于比较指令

统一化指令

语法	操作	编译结果

cmphs rx, ry	RX与RY作无符号比较。
	If RX >= RY, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令 cmphs16 rx, ry

说明： 将RX的值减去RY的值，结果与0作比较，并对C位进行更新。cmphs进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

异常： 无

16位指令

操作： RX与RY作无符号比较。

If RX >= RY, then

C ← 1;

else

C ← 0;

语法： cmphs16 rx, ry

说明： 将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmphs16 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于 RY，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 1	RY	RX	0 0

CMPHSI——立即数无符号大于等于比较指令

统一化指令

语法	操作	编译结果

cmphsi rx, oimm16	RX与立即数作无符号比较。
	If RX >= zero_extend(OIMM16),
	C ← 1;
	else
	C ← 0;
		根据立即数和寄存器的范围编译为对应的16位或32位指令。
		if (oimm16<33) and (x<8),then
		cmphsi16 rx, oimm5;
		else
		cmphsi32 rx, oimm16;

说明： 将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM16，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 立即数的范围为 0x1-0x10000。

异常： 无

16位指令

操作： RX与立即数作无符号比较。

If RX >= zero_extend(OIMM5), then

C ← 1;

else

C ← 0;

语法： cmphsi16 rx, oimm5

说明： 将带偏置 1 的 5 位立即数（OIMM5）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmphsi16 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM5，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。

注意：二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： 根据比较结果设置条件位 C

限制： 寄存器的范围为 r0-r7；立即数的范围为 1-32。

异常： 无

指令格式：

15	14 10	8 7	5 4	0
0	0 1 1 1	RX	0 0 0	IMM5

IMM5 域——指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000——与 1 比较

00001——与 2 比较

……

11111——与 32 比较

32位指令

操作： RX与立即数作无符号比较。

```
If RX >= zero_extend(OIMM16), then
   C ← 1;
else
   C ← 0;
```

语法： cmphsi32 rx, oimm16

说明： 将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该32位值，结果与0作比较，并对C位进行更新。cmphsi32 进行无符号比较，即操作数被认为是无符号数。如果 RX 大于等于零扩展后的 OIMM16，即减法结果大于等于 0，则设置条件位 C；否则，清除条件位 C。

注意：二进制操作数 IMM16 等于 OIMM16 - 1。

影响标志位： 根据比较结果设置条件位 C

限制： 立即数的范围为 0x1-0x10000。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0		1 1 0 0 0		RX		IMM16	

IMM16 域——指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM16 比起二进制操作数 IMM16 需偏置 1。

0000000000000000——与 0x1 比较

0000000000000001——与 0x2 比较

……

1111111111111111——与 0x10000 比较

CMPLT——有符号小于比较指令

统一化指令

语法	操作	编译结果

cmplt rx, ry	RX与RY作有符号比较。
	If RX < RY, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令 cmplt16 rx, ry

说明： 将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

异常： 无

16位指令

操作： RX与RY作有符号比较。

If RX < RY, then

C ← 1;

else

C ← 0;

语法： cmplt16 rx, ry

说明： 将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。cmplt16 进行有符号比较，即操作数被认为是补码形式的有符号数。如果 RX 小于 RY，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 0 1	RY	RX	0 1

CMPLTI——立即数有符号小于比较指令

统一化指令

语法	操作	编译结果

cmplti rx, oimm16	RX与立即数作有符号比较。
	If RX < zero_extend(OIMM16),
	C ← 1;
	else
	C ← 0;
		根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<8) and (oimm16<33), then
		cmplti16 rx, oimm5;
		else
		cmplti32 rx, oimm16;

说明： 将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmplti 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM16，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 立即数的范围为 0x1-0x10000。

异常： 无

16位指令

操作：
RX与立即数作有符号比较。
If RX < zero_extend(OIMM5), then
C ← 1;
else
C ← 0;

语法： cmplti16 rx, oimm5

说明： 将带偏置 1 的 5 位立即数（OIMM5）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmplti16 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM5，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。

注意：二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： 根据比较结果设置条件位 C

限制： 寄存器的范围为 r0-r7；立即数的范围为 1-32。

异常： 无

指令格式：

15	14 10	8 7	5 4	0
0	0 1 1 1	RX	0 0 1	IMM5

IMM5 域——指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000——与 1 比较

00001——与 2 比较

……

11111——与 32 比较

32位指令

操作： RX与立即数作有符号比较。

If RX < zero_extend(OIMM16), then

C ← 1;

else

C ← 0;

语法： cmplti32 rx, oimm16

说明： 将带偏置 1 的 16 位立即数（OIMM16）零扩展至 32 位，然后用 RX 的值减去该 32 位值，结果与 0 作比较，并对 C 位进行更新。cmplti32 进行有符号比较，即 RX 的值被认为是补码形式的有符号数。如果 RX 小于零扩展后的 OIMM16，即减法结果小于 0，则设置条件位 C；否则，清除条件位 C。

注意：二进制操作数 IMM16 等于 OIMM16 - 1。

影响标志位： 根据比较结果设置条件位 C

限制： 立即数的范围为 0x1-0x10000。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 1 0 0 1	RX	IMM16

IMM16 域——指定不带偏置立即数的值。

注意：参与比较的立即数 OIMM16 比起二进制操作数 IMM16 需偏置 1。

0000000000000000——与 0x1 比较

0000000000000001——与 0x2 比较

……

1111111111111111——与 0x10000 比较

CMPNE——不等比较指令

统一化指令

语法	操作	编译结果

cmpne rx, ry	RX与RY作比较。
	If RX != RY, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令
		cmpne16 rx, ry

说明： 将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

异常： 无

16位指令

操作： RX与RY作比较。

If RX != RY, then

C ← 1;

else

C ← 0;

语法： cmpne16 rx, ry

说明： 将 RX 的值减去 RY 的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于 RY，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 0 1	RY	RX	1 0

CMPNEI——立即数不等比较指令

统一化指令

语法	操作	编译结果

cmpnei rx, imm16	RX与立即数作比较。
	If RX != zero_extend(imm16),
	C ← 1;
	else
	C ← 0;
		根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<8) and (imm16<33), then
		cmpnei16 rx, imm5;
		else
		cmpnei32 rx, imm16;

说明： 将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

16位指令

操作：
RX与立即数作比较。
If RX != zero_extend(IMM5), then
C ← 1;
else
C ← 0;

语法： cmpnei16 rx, imm5

说明： 将 RX 的值减去零扩展至 32 位的 5 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM5，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 寄存器的范围为 r0-r7；
立即数的范围为 0-31。

异常： 无

指令格式：

15 14 10 8 7 5 4 0

0	0 1 1 1	RX	0 1 0	IMM5

32位指令

操作： RX与立即数作比较。

If RX != zero_extend(imm16), then

C ← 1;

else

C ← 0;

语法： cmpnei rx, imm16

说明： 将 RX 的值减去零扩展至 32 位的 16 位立即数的值，结果与 0 作比较，并对 C 位进行更新。如果 RX 不等于零扩展后的 IMM16，即减法结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据比较结果设置条件位 C

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	0
1	1 1 0 1 0		1 1 0 1 0	RX		IMM16		

DECF——C 为 0 立即数减法指令

统一化指令

语法	操作	编译结果

decf rz, rx, imm5	if C==0, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;
		仅存在 32 位指令
		decf32 rz, rx, imm5

说明： 如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作：
if C==0, then
RZ ← RX - zero_extend(IMM5);
else
RZ ← RZ;

语法： decf32 rz, rx, imm5

说明： 如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 1 0 0	IMM5

DECT——C 为 1 立即数减法指令

统一化指令

语法	操作	编译结果

dect rz, rx, imm5	if C==1, then
	RZ ← RX - zero_extend(IMM5);
	else
	RZ ← RZ;
		仅存在 32 位指令
		dect32 rz, rx, imm5

说明： 如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作：
if C==1, then
RZ ← RX - zero_extend(IMM5);
else
RZ ← RZ;

语法： dect32 rz, rx, imm5

说明： 如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用 RX 的值减去该 32 位值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 1 0 0 0	IMM5

DOZE——进入低功耗睡眠模式指令

统一化指令

语法	操作	编译结果
doze	进入低功耗睡眠模式	仅存在 32 位指令 doze32

说明： 此指令使处理器进入低功耗睡眠模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，相应的外围设备也被停止。

影响标志位： 不影响

异常： 特权违反异常

32位指令

操作： 进入低功耗睡眠模式

语法： doze32

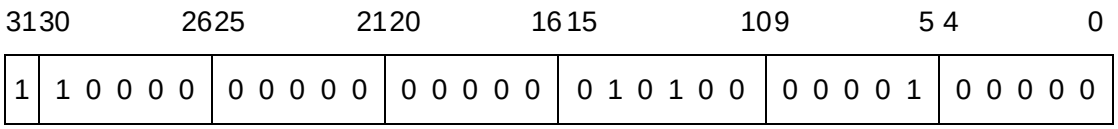
属性： 特权指令

说明： 此指令使处理器进入低功耗睡眠模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，相应的外围设备也被停止。

影响标志位： 不影响

异常： 特权违反异常

指令格式：



FF0——快速找 0 指令

统一化指令

语法	操作	编译结果
ff0 rz, rx	RZ ← find_first_0(RX);	仅存在 32 位指令 ff0.32 rz, rx

说明： 查找 RX 第一个为 0 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 0，返回 RZ 的值为 0。如果在 RX 没有为 0 的位，返回 RZ 的值为 32。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← find_first_0(RX);

语法： ff0.32 rz, rx

说明： 查找 RX 第一个为 0 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 0，返回 RZ 的值为 0。如果在 RX 没有为 0 的位，返回 RZ 的值为 32。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 1 1	0 0 0 0 1	RZ

FF1——快速找 1 指令

统一化指令

语法	操作	编译结果
ff1 rz, rx	RZ ← find_first_1(RX);	仅存在 32 位指令 ff1.32 rz, rx

说明： 查找 RX 第一个为 1 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 1，返回 RZ 的值为 0。如果在 RX 没有为 1 的位，返回 RZ 的值为 32。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← find_first_1(RX);

语法： ff1.32 rz, rx

说明： 查找 RX 第一个为 1 的位，并把查找结果返回到 RZ。查找顺序是从 RX 的最高位扫描到最低位。如果 RX 的最高位（RX[31]）为 1，返回 RZ 的值为 0。如果在 RX 没有为 1 的位，返回 RZ 的值为 32。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 1 1	0 0 0 1 0	RZ

GRS——符号产生指令

统一化指令

语法	操作	编译结果
grs rz, label grs rz, imm32	RZ ← PC + sign_extend(offset << 1);	仅存在 32 位指令。 grs32 rz, label grs32 rz, imm32

说明： 产生符号的值，该值由 label 所在位置，或 32 位立即数（IMM32）确定。符号的值由当前程序 PC 加上左移 1 位的 18 位相对偏移量有符号扩展到 32 位后的值得到。符号的值的有效范围是±256KB 地址空间。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← PC + sign_extend(offset << 1);

语法： grs rz, label

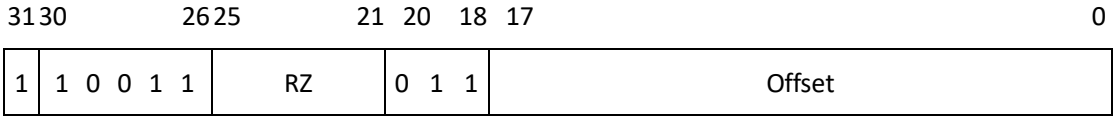
grs rz, imm32

说明： 产生符号的值，该值由 label 所在位置，或 32 位立即数（IMM32）确定。符号的值由当前程序 PC 加上左移 1 位的 18 位相对偏移量有符号扩展到32位后的值得到。符号的值的有效范围是±256KB地址空间。

影响标志位： 无影响

异常： 无

指令格式：



INCF——C 为 0 立即数加法指令

统一化指令

语法	操作	编译结果

incf rz, rx, imm5	if C==0, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;
		仅存在 32 位指令
		incf32 rz, rx, imm5

说明： 如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作： if C==0, then
RZ ← RX + zero_extend(IMM5);
else
RZ ← RZ;

语法： incf32 rz, rx, imm5

说明： 如果条件位 C 为 0，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RZ		RX		0 0 0 0 1 1		0 0 0 0 1		IMM5	

INCT——C 为 1 立即数加法指令

统一化指令

语法	操作	编译结果

inct rz, rx, imm5	if C==1, then
	RZ ← RX + zero_extend(IMM5);
	else
	RZ ← RZ;
		仅存在 32 位指令
		inct32 rz, rx, imm5

说明： 如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作： if C==1, then
RZ ← RX + zero_extend(IMM5);
else
RZ ← RZ;

语法： inct32 rz, rx, imm5

说明： 如果条件位 C 为 1，将 5 位立即数零扩展至 32 位，用该 32 位值加 RX 的值，把结果存在 RZ；否则，RZ 与 RX 的值不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RZ		RX		0 0 0 0 1 1		0 0 0 1 0		IMM5	

IPUSH——中断压栈指令

统一化指令

语法	操作	编译结果
ipush	将中断的通用寄存器现场{R0~R3, R12, R13 }存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端； MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0}; SP←SP-24;	仅存在 16 位指令 ipush16

说明： 将中断的通用寄存器现场{ R0~R3, R12, R13 }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

16位指令

操作： 将中断的通用寄存器现场{R0~R3, R12, R13 }存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端；
MEM[SP-4]~MEM[SP-24] ←{R13,R12,R3~R0};
SP←SP-24;

语法： IPUSH16

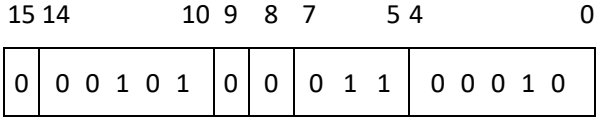
属性 无

说明： 将中断的通用寄存器现场{ R0~R3, R12, R13 }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

指令格式：



IPOP——中断出栈指令

统一化指令

语法	操作	编译结果
ipop	从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端； {R0~R3,R12,R13} ← MEM[SP]~MEM[SP+20]; SP←SP+24;	仅存在 16 位指令 ipop16

说明： 从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

16位指令

操作： 从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端；
{R0~R3,R12,R13} ← MEM[SP]~MEM[SP+20];
SP←SP+24;

语法： IPOP16

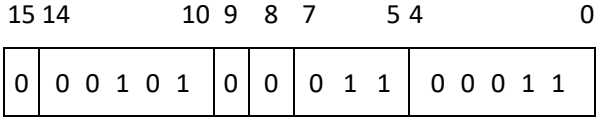
属性 无

说明： 从堆栈指针寄存器指向堆栈中载入中断的通用寄存器现场{R0~R3, R12, R13 }，然后更新堆栈指针寄存器到堆栈存储器的顶端。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常

指令格式：



IXH——索引半字指令

统一化指令

语法	操作	编译结果
ixh rz, rx, ry	RZ ← RX + (RY << 1)	仅存在 32 位指令 ixh32 rz, rx, ry

说明： 将 RY 的值左移一位后加上 RX 的值，并把结果存入 RZ。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← RX + (RY << 1)

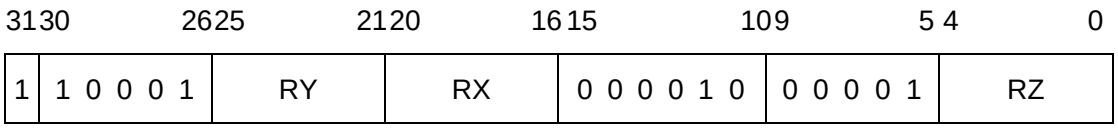
语法： ixh32 rz, rx, ry

说明： 将 RY 的值左移一位后加上 RX 的值，并把结果存入 RZ。

影响标志位： 无影响

异常： 无

指令格式：



IXW——索引字指令

统一化指令

语法	操作	编译结果
ixw rz, rx, ry	RZ ← RX + (RY << 2)	仅存在 32 位指令 ixw32 rz, rx, ry

说明： 将 RY 的值左移两位后加上 RX 的值，并把结果存入 RZ。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← RX + (RY << 2)

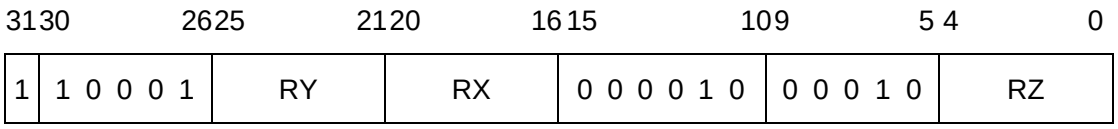
语法： ixw32 rz, rx, ry

说明： 将 RY 的值左移两位后加上 RX 的值，并把结果存入 RZ。

影响标志位： 无影响

异常： 无

指令格式：



JMP——寄存器跳转指令

统一化指令

语法	操作	编译结果
jmp rx	跳转到寄存器指定的位置 PC ← RX & 0xfffffffe	仅存在 16 位指令。 jmp16 rx

说明： 程序跳转到寄存器 RX 指定的位置，RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。

影响标志位： 无影响

异常： 无

16位指令

操作： 跳转到寄存器指定的位置

PC ← RX & 0xfffffffe

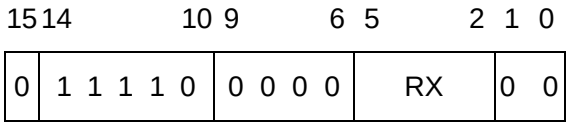
语法： jmp16 rx

说明： 程序跳转到寄存器 RX 指定的位置，RX 的最低位被忽略。JMP 指令的跳转范围是全部 4GB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：



JMPIX——寄存器索引跳转指令

统一化指令

语法	操作	编译结果
jmpix rx, imm	跳转到寄存器索引指定的位置 PC ← SVBR + (RX & 0xff) * IMM	仅存在 16 位指令。 jmpix16 rx, imm;

说明： 程序跳转到 SVBR + RX[7:0] * IMM 的位置，IMM ∈ {16, 24, 32, 40}。RX 的高 24 位被忽略。

影响标志位： 无影响

异常： 无

注意： 该指令仅实现于支持二进制代码转译机制的S802处理器。

16位指令

操作： 跳转到寄存器索引指定的位置

PC ← SVBR + (RX & 0xff) * IMM

语法： jmpix16 rx, imm

说明： 程序跳转到 SVBR + RX[7:0] * IMM 的位置，IMM ∈ {16, 24, 32, 40}。RX 的高 24 位被忽略。

影响标志位： 无影响

异常： 无

指令格式：

15	14 11	10 8	7 2	0
0	0 1 1 1	RX	1 1 1 0 0 0	IMM2

IMM2 域——指定立即数的值。

注意：二进制编码的 IMM2 值与此跳转指令中 IMM 值的对应关系如下：

2’b00——乘 16

2’b01——乘 24

2’b10——乘 32

2’b11——乘 40

JSR——寄存器跳转到子程序指令

统一化指令

语法	操作	编译结果
jsr rx	链接并跳转到寄存器指定的子程序位置 R15 ← PC + 4， PC ← RX & 0xfffffffe	仅存在 16 位指令。 jsr16 rx

说明： 子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。

影响标志位： 无影响

异常： 无

16位指令

操作： 链接并跳转到寄存器指定的子程序位置

R15 ← PC + 4，PC ← RX & 0xfffffffe

语法： jsr16 rx

说明： 子程序寄存器跳转，将子程序的返回地址（下一条指令的 PC，即当前 PC+4）保存在链接寄存器 R15 中，程序跳转到寄存器 RX 的内容指定的子程序位置处执行，RX 的最低位被忽略。JSR 指令的跳转范围是全部 4GB 地址空间。

影响标志位： 无影响

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 1 0	0 0 0 0	RX	0 1

LD.B——无符号扩展字节加载指令

统一化指令

语法	操作	编译结果

ld.b rz,(rx, disp)	RZ ← zero_extend(MEM[RX + zero_extend(offset)])	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (disp<32)and(x<7) and (z<7),
		then
		ld16.b rz, (rx, disp);
		else
		ld32.b rz, (rx, disp);

说明： 从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.B 指令可以寻址+4KB 地址空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16位指令

操作： 从存储器加载字节到寄存器，无符号扩展

RZ ← zero_extend(MEM[RX + zero_extend(offset)])

语法： ld16.b rz, (rx, disp)

说明： 从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.B 指令可以寻址+32B 的地址空间。

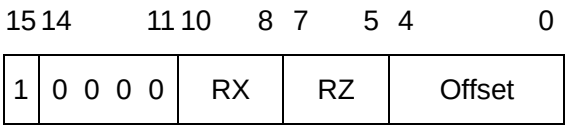
注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：



32位指令

操作： 从存储器加载字节到寄存器，无符号扩展

RZ ← zero_extend(MEM[RX + zero_extend(offset)])

语法： ld32.b rz, (rx, disp)

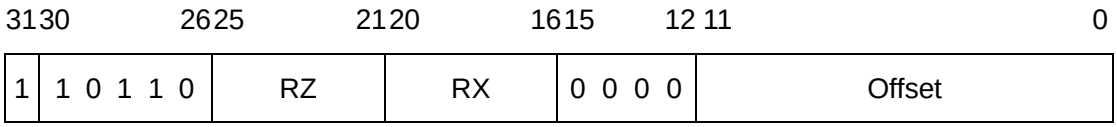
说明： 从存储器加载的字节经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.B 指令可以寻址+4KB 地址空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：



LD.BS——有符号扩展字节加载指令

统一化指令

语法	操作	编译结果
ld.bs rz, (rx, disp)	RZ ← sign_extend(MEM[RX + zero_extend(offset)])	仅存在 32 位指令。 ld32.bs rz, (rx, disp)

说明： 从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.BS 指令可以寻址+4KB 地址空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32位指令

操作： 从存储器加载字节到寄存器，有符号扩展

RZ ← sign_extend(MEM[RX + zero_extend(offset)])

语法： ld32.bs rz, (rx, disp)

说明： 从存储器加载的字节经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.BS 指令可以寻址+4KB 地址空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	1 0 1 1 0		RZ		RX	0 1 0 0			Offset	

LD.H——无符号扩展半字加载指令

统一化指令

语法	操作	编译结果

ld.h rz, (rx, disp)	RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (disp<64)and(x<7) and (z<7), then
		ld16.h rz, (rx, disp);
		else
		ld32.h rz, (rx, disp);

说明： 从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.H 指令可以寻址+8KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16位指令

操作： 从存储器加载半字到寄存器，无符号扩展

RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])

语法： ld16.h rz, (rx, disp)

说明： 从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.H 指令可以寻址+64B 的地址空间。

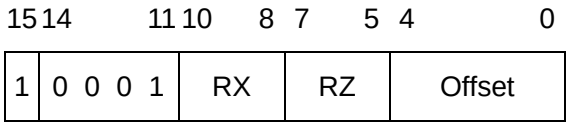
注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：



32位指令

操作： 从存储器加载半字到寄存器，无符号扩展

RZ ← zero_extend(MEM[RX + zero_extend(offset << 1)])

语法： ld32.h rz, (rx, disp)

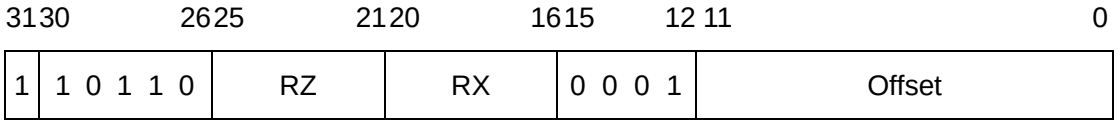
说明： 从存储器加载的半字经零扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.H 指令可以寻址+8KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：



LD.HS——有符号扩展半字加载指令

统一化指令

语法	操作	编译结果
ld.hs rz, (rx, disp)	RZ ← sign_extend(MEM[RX + zero_extend(offset << 1)])	仅存在 32 位指令。 ld32.hs rz, (rx, disp)

说明： 从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.HS 指令可以寻址+8KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32位指令

操作： 从存储器加载半字到寄存器，有符号扩展

RZ ← sign_extend(MEM[RX + zero_extend(offset << 1)])

语法： ld32.hs rz, (rx, disp)

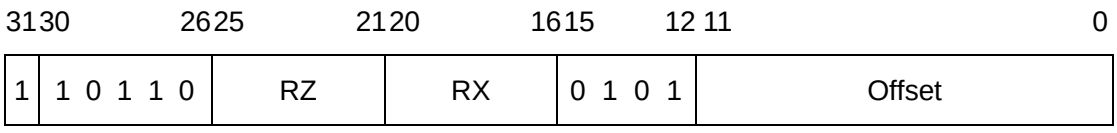
说明： 从存储器加载的半字经有符号扩展到 32 位后存放于寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.HS 指令可以寻址+8KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：



LD.W——字加载指令

统一化指令

语法	操作	编译结果

ld.w rz, (rx, disp)	RZ ← MEM[RX + zero_extend(offset << 2)]	根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x=sp) and (z<7) and (disp < 1024),
		ld16.w rz, (sp, disp);
		else if (disp<128) and (x<7) and (z<7),
		ld16.w rz, (rx, disp);
		else
		ld32.w rz, (rx, disp);

说明： 从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD.W 指令可以寻址+16KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16位指令

操作： 从存储器加载字到寄存器

RZ ← MEM[RX + sign_extend(offset << 2)]

语法： ld16.w rz, (rx, disp)

ld16.w rz, (sp, disp)

说明： 从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。当 RX 为 SP 时，存储器的有效地址由基址寄存器 RX 加上左移 2 位的 8 位相对偏移量无符号扩展到 32 位后的值得到。当 rx 为其它寄存器时，存储器的有效地址由基址寄存器 RX 加上左移 2 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。LD16.W 指令可以寻址+1KB 的地址空间。

注意，偏移量 DISP 是二进制操作数 IMM5 左移两位得到的。当基址寄存器 RX 为 SP 时，偏移量 DISP 是二进制操作数{IMM3, IMM5}左移两位得到的。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

ld16.w rz, (rx, disp)

15	14 11	10 8	7 5	4 0
1	0 0 1 0	RX	RZ	IMM5

ld16.w rz, (sp, disp)

15	14 11	10 8	7 5	4 0
1	0 0 1 1	IMM3	RZ	IMM5

32位指令

操作： 从存储器加载字到寄存器

RZ ← MEM[RX + zero_extend(offset << 2)]

语法： ld32.w rz, (rx, disp)

说明： 从存储器加载字到寄存器 RZ 中。采用寄存器加立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。LD32.W 指令可以寻址+16KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

31	30	26	25	21	20	16	15	12	11	0
1	1 0 1 1 0		RZ		RX	0 0 1 0			Offset	

LDM——连续多字加载指令

统一化指令

语法	操作	编译结果

ldm ry-rz, (rx)	从存储器加载连续的多个字到一片连续的寄存器堆中
	dst ← Y; addr ← RX;
	for (n = 0; n <= (Z-Y); n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}
		仅存在 32 位指令。
		ldm32 ry-rz, (rx);

说明： 从存储器依次加载连续的多个字到寄存器 RY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字加载到寄存器 RY 中，第二个字加载到寄存器 RY+1 中，依次类推，最后一个字加载到寄存器 RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。

影响标志位： 无影响

限制： RZ 应当大于等于 RY。

RY-RZ 范围内不应该包含基址寄存器 RX，否则结果不可预测。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32位指令

操作： 从存储器加载连续的多个字到一片连续的寄存器堆中

```
dst ← Y; addr ← RX;
for (n = 0; n <= IMM5; n++){
   Rdst ← MEM[addr];
   dst ← dst + 1;
   addr ← addr + 4;
}
```

语法： ldm32 ry-rz, (rx)

说明： 从存储器依次加载连续的多个字到寄存器 RY 开始的一片连续寄存器堆中，即将存储器指定地址开始的第一个字加载到寄存器 RY 中，第二个字加载到寄存器 RY+1 中，依次类推，最后一个字加载到寄存器 RZ 中。存储器的有效地址由基址寄存器 RX 的内容决定。

影响标志位： 无影响

限制： RZ 应当大于等于 RY。

RY-RZ 范围内不应该包含基址寄存器 RX，否则结果不可预测。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 1 0 0		RY		RX		0 0 0 1 1 1		0 0 0 0 1		IMM5	

IMM5 域——指定目标寄存器的个数，IMM5 = Z - Y。

00000——1 个目的寄存器

00001——2 个目的寄存器

……

11111——32 个目的寄存器

LDQ——连续四字加载指令#

统一化指令

语法	操作	编译结果

ldq r4-r7, (rx)	从存储器加载连续的四个字到寄存器 R4—R7 中
	dst ← 4; addr ← RX;
	for (n = 0; n <= 3; n++){
	Rdst ← MEM[addr];
	dst ← dst + 1;
	addr ← addr + 4;
	}
		仅存在 32 位指令。
		ldq32 r4-r7, (rx);

说明： 从存储器依次加载连续的 4 个字到寄存器堆[R4, R7]（包括边界）中，即将存储器指定地址开始的第一个字加载到寄存器 R4 中，第二个字加载到寄存器 R5 中，第三个字加载到寄存器 R6 中，第四个字加载到寄存器 R7 中。存储器的有效地址由基址寄存器 RX 的内容决定。注意，该指令是 ldm r4-r7, (rx)的伪指令。

影响标志位： 无影响

限制： R4-R7 范围内不应该包含基址寄存器 RX，否则结果不可预测。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

32位指令

操作： 从存储器加载连续的四个字到寄存器 R4—R7 中

```
dst ← 4; addr ← RX;
for (n = 0; n <= 3; n++){
   Rdst ← MEM[addr];
   dst ← dst + 1;
   addr ← addr + 4;
}
```

语法： ldq32 r4-r7, (rx)

说明： 从存储器依次加载连续的 4 个字到寄存器堆[R4, R7]（包括边界）中，即将存储器指定地址开始的第一个字加载到寄存器 R4 中，第二个字加载到寄存器 R5 中，第三个字加载到寄存器 R6 中，第四个字加载到寄存器 R7 中。存储器的有效地址由基址寄存器 RX 的内容决定。

注意，该指令是 ldm32 r4-r7, (rx)的伪指令。

影响标志位： 无影响

限制： R4-R7 范围内不应该包含基址寄存器 RX，否则结果不可预测。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 1 0 0	0 0 1 0 0	RX	0 0 0 1 1 1	0 0 0 0 1	0 0 0 1 1

LRW——存储器读入指令

统一化指令

语法	操作	编译结果

lrw rz, label
lrw rz, imm32
	从存储器加载字到寄存器
	RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])
		根据加载的范围编译为对应的 16 位或 32 位指令
		if(offset<512B), then
		lrw16 label;
		lrw16 imm32;
		else
		lrw32 label;
		lrw32 imm32;

说明： 加载 label 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ。存储器地址根据 PC 加左移两位的相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

16位指令

操作： 从存储器加载字到寄存器

RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])

语法： lrw16 rz, label

lrw16 rz, imm32

说明： 加载 label 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ。存储器地址根据 PC 加左移两位的 10 位相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。

注意，若 IMM1 为 1，相对偏移量 Offset 等于二进制编码{0，{IMM2，IMM5}}。

若 IMM1 为 0， 相对偏移量 Offset 等于二进制编码{1，{！{IMM2，IMM5}}}。

IMM1,IMM2,IMM5,RZ 同时为零则该指令为 BKPT 指令，并非 LRW16。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

15	14 13	12	11 10	9 8	7 5	4 0
0	00	IMM1	0 0	IMM2	RZ	IMM5

32位指令

操作： 从存储器加载字到寄存器

RZ ← zero_extend(MEM[(PC + zero_extend(offset << 2)) & 0xfffffffc])

语法： lrw32 rz, label

lrw32 rz, imm32

说明： 加载 label 所在位置的字，或 32 位立即数（IMM32）至目的寄存器 RZ。存储器地址根据 PC 加左移两位的 16 位相对偏移量，并无符号扩展到 32 位后，再经最低两位强制清零得到。LRW 指令的加载范围是全部 4GB 地址空间。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

31	30 26	25 21	20 16	15 0
1	1 1 0 1 0	1 0 1 0 0	RZ	Offset

LSL——逻辑左移指令

统一化指令

语法	操作	编译结果

lsl rz, rx	RZ ← RZ << RX[5:0]	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		lsl16 rz, rx;
		else
		lsl32 rz, rz, rx;

lsl rz, rx, ry	RZ ← RX << RY[5:0]	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		lsl16 rz, ry
		else
		lsl32 rz, rx, ry

说明： 对于 lsl rz, rx，将 RZ 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0]的值大于 31，那么 RZ 将被清零；

对于 lsl rz, rx, ry，将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ << RX[5:0]

语法： lsl16 rz, rx

说明： 将 RZ 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 0	RZ	RX	0 0

32位指令

操作： RZ ← RX << RY[5:0]

语法： lsl32 rz, rx, ry

说明： 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 0 0 1	RZ

LSLC——立即数逻辑左移至 C 位指令

统一化指令

语法	操作	编译结果
lslc rz, rx, oimm5	RZ ← RX << OIMM5, C ← RX[32 – OIMM5]	仅存在 32 位指令。 lslc32 rz, rx, oimm5

说明： 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最低位，RZ 被清零。

影响标志位： C ← RX[32 – OIMM5]

限制： 立即数的范围为 1-32。

异常： 无

32位指令

操作： RZ ← RX << OIMM5, C ← RX[32 – OIMM5]

语法： lslc32 rz, rx, oimm5

说明： 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，左移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最低位，RZ 被清零。

注意：二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： C ← RX[32 – OIMM5]

限制： 立即数的范围为 1-32。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 1		0 0 0 0 1		RZ	

IMM5 域——指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000——移 1 位

00001——移 2 位

……

11111——移 32 位

LSLI——立即数逻辑左移指令

统一化指令

语法	操作	编译结果

lsli rz, rx, imm5	RZ ← RX << IMM5	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<8) and (z<8), then
		lsli16 rz, rx, imm5
		else
		lsli32 rz, rx, imm5

说明： 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

16位指令

操作： RZ ← RX << IMM5

语法： lsli16 rz, rx, imm5

说明： 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；

立即数的范围为 0-31。

异常： 无

指令格式：

15 14 11 10 8 7 5 4 0

0	1 0 0 0	RX	RZ	IMM5

32位指令

操作： RZ ← RX << IMM5

语法： lsli32 rz, rx, imm5

说明： 将 RX 的值进行逻辑左移（原值左移，右侧移入 0），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 0 0 0 1	RZ

LSR——逻辑右移指令

统一化指令

语法	操作	编译结果

lsr rz, rx	RZ ← RZ >> RX[5:0]	根据寄存器的范围编译为对应的16 位或 32 位指令。
		if (z<16) and (x<16), then
		lsr16 rz, rx;
		else
		lsr32 rz, rz, rx;

lsr rz, rx, ry	RZ ← RX >> RY[5:0]	根据寄存器的范围编译为对应的16 位或 32 位指令。
		if (x==z) and (z<16) and (y<16), then
		lsr16 rz, ry;
		else
		lsr32 rz, rx, ry;

说明： 对于 lsr rz, rx，将 RZ 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0]的值大于 31，那么 RZ 将被清零。

对于 lsr rz, rx, ry，将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ >> RX[5:0]

语法： lsr16 rz, rx

说明： 将 RZ 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RX 低 6 位（RX[5:0]）的值确定；如果 RX[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 0	RZ	RX	0 1

32位指令

操作： RZ ← RX >> RY[5:0]

语法： lsr32 rz, rx, ry

说明： 将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 RY 低 6 位（RY[5:0]）的值确定；如果 RY[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 0 0 1 0	RZ

LSRC——立即数逻辑右移至 C 位指令

统一化指令

语法	操作	编译结果
lsrc rz, rx, oimm5	RZ ← RX >> OIMM5, C ← RX[OIMM5 - 1]	仅存在 32 位指令。 lsrc32 rz, rx, oimm5

说明： 将 RX 的值进行逻辑右移（原值右移，左侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位，RZ 被清零。

影响标志位： C ← RX[OIMM5 - 1]

限制： 立即数的范围为 1-32。

异常： 无

32位指令

操作： RZ ← RX >> OIMM5, C ← RX[OIMM5 - 1]

语法： lsrc32 rz, rx, oimm5

说明： 将 RX 的值进行逻辑右移（原值右移，左侧移入 0），把移出最末位存入条件位 C，移位结果存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位，RZ 被清零。

注意：二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： C ← RX[OIMM5 - 1]

限制： 立即数的范围为 1-32。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		IMM5		RX		0 1 0 0 1 1		0 0 0 1 0		RZ	

IMM5 域——指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000——移 1 位

00001——移 2 位

……

11111——移 32 位

LSRI——立即数逻辑右移指令

统一化指令

语法	操作	编译结果

lsri rz, rx, imm5	RZ ← RX >> IMM5	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<8) and (z<8), then
		lsri16 rz, rx, imm5
		else
		lsri32 rz, rx, imm5

说明： 将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值不变或者将与 RX 相同。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

16位指令

操作： RZ ← RX >> IMM5

语法： lsri16 rz, rx, imm5

说明： 将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将不变。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 0-31。

异常： 无

指令格式：

15 14 11 10 8 7 5 4 0

0	1 0 0 1	RX	RZ	IMM5

32位指令

操作： RZ ← RX >> IMM5

语法： lsri32 rz, rx, imm5

说明： 将 RX 的值进行逻辑右移（原值右移，左侧移入 0），结果存入 RZ，右移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 0	0 0 0 1 0	RZ

MFCR——控制寄存器读传送指令

统一化指令

语法	操作	编译结果
mfcr rz, cr<x, sel>	将控制寄存器的内容传送到通用寄存器中 RZ ← CR<X, sel>	仅存在 32 位指令。 mfcr32 rz, cr<x, sel>

属性： 特权指令

说明： 将控制寄存器 CR<x, sel>的内容传送到通用寄存器 RZ 中。

影响标志位： 无影响

异常： 特权违反异常

32位指令

操作： 将控制寄存器的内容传送到通用寄存器中

RZ ← CR<X, sel>

语法： mfcr32 rz, cr<x, sel>

属性： 特权指令

说明： 将控制寄存器 CR<x, sel>的内容传送到通用寄存器 RZ 中。

影响标志位： 无影响

异常： 特权违反异常

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
1	1 0 0 0 0	sel	CRX	0 1 1 0 0 0	0 0 0 0 1	RZ

MOV——数据传送指令#

统一化指令

语法	操作	编译结果
mov rz, rx	RZ ← RX	总是编译为 16 位指令。 mov16 rz, rx

说明： 把 RX 中的值复制到目的寄存器 RZ 中。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RX

语法： mov16 rz, rx

说明： 把 RX 中的值复制到目的寄存器 RZ 中。

注意，该指令寄存器索引范围为 r0-r31。

影响标志位： 无影响

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 0 1 1	RZ	RX	1 1

32位指令

操作： RZ ← RX

语法： mov32 rz, rx

说明： 把 RX 中的值复制到目的寄存器 RZ 中。

注意，该指令是 lsli32 rz, rx, 0x0 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：

31 30 26 25 21 20 16 15 10 9 5 4 0

1	1 0 0 0 1	0 0 0 0 0	RX	0 1 0 0 1 0	0 0 0 0 1	RZ

MOVF——C 为 0 数据传送指令#

统一化指令

语法	操作	编译结果

movf rz, rx	if C==0, then
	RZ ← RX;
	else
	RZ ← RZ;
		仅存在 32 位指令。
		movf32 rz, rx

说明： 如果 C 为 0，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。

注意，该指令是 incf rz, rx, 0x0 的伪指令。

影响标志位： 无影响

异常： 无

32位指令

操作：
if C==0, then
RZ ← RX;
else
RZ ← RZ;

语法： movf32 rz, rx

说明： 如果 C 为 0，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。

注意，该指令是 incf32 rz, rx, 0x0 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RZ		RX		0 0 0 0 1 1		0 0 0 0 1		0 0 0 0 0	

MOVI——立即数数据传送指令

统一化指令

语法	操作	编译结果

movi rz, imm	RZ ← zero_extend(IMM);	根据立即数和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (imm16<256) and (z<7), then
		movi16 rz, imm8;
		else
		movi32 rz, imm16;

说明： 将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

16位指令

操作： RZ ← zero_extend(IMM8);

语法： movi16 rz, imm8

说明： 将 8 位立即数零扩展至 32 位，然后传送至目的 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 0-255。

异常： 无

15 14 11 10 8 7 0

0	0 1 1 0	RZ	IMM8

32位指令

操作： RZ ← zero_extend(IMM16);

语法： movi32 rz, imm16

说明： 将 16 位立即数零扩展至 32 位，然后传送至目的 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

31 30 26 25 21 20 16 15 0

1	1 1 0 1 0	1 0 0 0 0	RZ	IMM16

MOVIH——立即数高位数据传送指令

统一化指令

语法	操作	编译结果
movih rz, imm16	RZ ← zero_extend(IMM16) << 16	仅存在 32 位指令。 movih32 rz, imm16

说明： 将 16 位立即数零扩展至 32 位，然后逻辑左移 16 位，传送结果至目的 RZ。

该指令可配合 ori rz, imm16 指令产生任意 32 位立即数。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

32位指令

操作： RZ ← zero_extend(IMM16) << 16

语法： movih32 rz, imm16

说明： 将 16 位立即数零扩展至 32 位，然后逻辑左移 16 位，传送结果至目的 RZ。

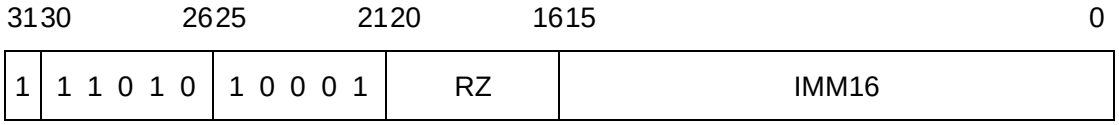
该指令可配合 ori32 rz, imm16 指令产生任意 32 位立即数。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

指令格式：



MOVT——C 为 1 数据传送指令#

统一化指令

语法	操作	编译结果

movt rz, rx	if C==1, then
	RZ ← RX;
	else
	RZ ← RZ;
		仅存在 32 位指令。
		movt32 rz, rx

说明： 如果 C 为 1，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。

注意，该指令是 inct rz, rx, 0x0 的伪指令。

影响标志位： 无影响

异常： 无

32位指令

操作：
if C==1, then
RZ ← RX;
else
RZ ← RZ;

语法： movt32 rz, rx

说明： 如果 C 为 1，把 RX 的值复制到目的寄存器 RZ 中；否则，RZ 的值不变。

注意，该指令是 inct32 rz, rx, 0x0 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RZ	RX	0 0 0 0 1 1	0 0 0 1 0	0 0 0 0 0

MTCR——控制寄存器写传送指令

统一化指令

语法	操作	编译结果
mtcr rx, cr<z, sel>	将通用寄存器的内容传送到控制寄存器中 CR<Z, sel> ← RX	仅存在 32 位指令。 mtcr32 rx, cr<z, sel>

属性： 特权指令

说明： 将通用寄存器 RX 的内容传送到控制寄存器 CR<z, sel>中。

影响标志位： 如果目标控制寄存器不是 PSR，则该指令不会影响标志位。

异常： 特权违反异常

32位指令

操作： 将通用寄存器的内容传送到控制寄存器中

CR<Z, sel> ← RX

语法： mtcr32 rx, cr<z, sel>

属性： 特权指令

说明： 将通用寄存器 RX 的内容传送到控制寄存器 CR<z, sel>中。

影响标志位： 如果目标控制寄存器不是 PSR，则该指令不会影响标志位。

异常： 特权违反异常

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 0	sel	RX	0 1 1 0 0 1	0 0 0 0 1	CRZ

MULT——乘法指令

统一化指令

语法	操作	编译结果

mult rz, rx	两个数相乘，结果的低 32 位放入通用寄存器中
	RZ ← RX × RZ
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		mult16 rz, rx;
		else
		mult32 rz, rz, rx;

mult rz, rx, ry	两个数相乘，结果的低 32 位放入通用寄存器中
	RZ ← RX × RY
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16)and (z<16), then
		mult16 rz, rx;
		else
		mult32 rz, rx, ry;

说明： 将两个源寄存器的内容相乘后结果的低 32 位存放到目的寄存器中，结果的高 32 位舍弃。不管源操作数被认为是有符号数还是无符号数，结果都相同。

影响标志位： 无影响

异常： 无

16位指令

操作： 两个数相乘，结果的低 32 位放入通用寄存器中

RZ ← RX × RZ

语法： mult16 rz, rx

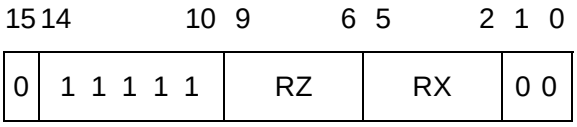
说明： 将通用寄存器 RX 和 RZ 的内容相乘后得到结果的低 32 位存放到通用寄存器 RZ 中，结果的高 32 位舍弃。不管源操作数被认为是有符号数还是无符号数，结果都相同。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：



32位指令

操作： 两个数相乘，结果的低 32 位放入通用寄存器中

RZ ← RX × RY

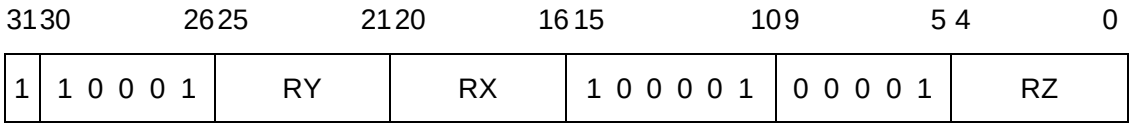
语法： mult32 rz, rx, ry

说明： 将通用寄存器 RX 和 RY 的内容相乘后结果的低 32 位存放到通用寄存器 RZ 中，结果的高 32 位舍弃。不管源操作数被认为是有符号数还是无符号数，结果都相同。

影响标志位： 无影响

异常： 无

指令格式：



MVC——C 位传送指令

统一化指令

语法	操作	编译结果
mvc rz	RZ ← C	仅存在 32 位指令。 mvc32 rz;

说明： 把条件位 C 传送到 RZ 的最低位，RZ 的其它位清零。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← C

语法： mvc32 rz

说明： 把条件位 C 传送到 RZ 的最低位，RZ 的其它位清零。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 1	0 1 0 0 0	RZ

MVCV——C 位取反传送指令

统一化指令

语法	操作	编译结果
mvcv rz	RZ ← (!C)	仅存在 16 位指令。 mvcv16 rz

说明： 把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← (!C)

语法： mvcv16 rz

说明： 把条件位 C 取反后传送到 RZ 的最低位，RZ 的其它位清零。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 0 1	RZ	0 0 0 0	1 1

NIE——中断嵌套使能指令

统一化指令

语法	操作	编译结果
nie	将中断的控制寄存器现场{EPSR, EPC}存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端，打开 PSR.IE 和 PSR.EE； MEM[SP-4] ←EPC; MEM[SP-8] ←EPSR; SP←SP-8; PSR({EE,IE}) ← 1	仅存在 16 位指令。 nie16

说明： 将中断的控制寄存器现场{EPSR, EPC }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈指针存储器的顶端，打开中断和异常使能位 PSR.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

16位指令

操作： 将中断的控制寄存器现场{EPSR, EPC}存储到堆栈存储器中，然后更新堆栈指针寄存器到堆栈存储器的顶端，打开 PSR.IE 和 PSR.EE；

MEM[SP-4] ←EPC;

MEM[SP-8] ←EPSR;

SP←SP-8;

PSR({EE,IE}) ← 1

语法： NIE16

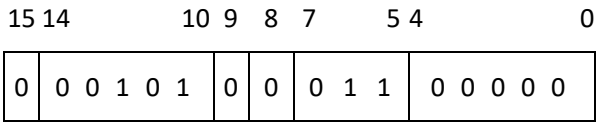
属性 特权指令

说明： 将中断的控制寄存器现场{EPSR, EPC }保存到堆栈存储器中，然后更新堆栈指针寄存器到堆栈指针存储器的顶端，打开中断和异常使能位 PSR.IE 和 PSR.EE。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

指令格式：



NIR——中断嵌套返回指令

统一化指令

语法	操作	编译结果
nir	从堆栈存储器中载入中断的控制寄存器现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回 EPSR←MEM[SP] EPC←MEM[SP+4]; SP←SP+8; PSR←EPSR; PC←EPC	仅存在 16 位指令。 nir16

说明： 从堆栈存储器中载入中断的现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

16位指令

操作： 从堆栈存储器中载入中断的控制寄存器现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；并中断返回

EPSR←MEM[SP]

EPC←MEM[SP+4];

SP←SP+8;

PSR←EPSR;

PC←EPC

语法： NIR16

属性 特权指令

说明： 从堆栈存储器中载入中断的现场到{EPSR, EPC }中，然后更新堆栈指针寄存器到堆栈存储器的顶端；PC 值恢复为控制寄存器 EPC 中的值，PSR 值恢复为 EPSR 的值，指令执行从新的 PC 地址处开始。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

异常： 访问错误异常、未对齐异常、特权违反异常

指令格式：

15	14 10	9	8	7 5	4 0
0	0 0 1 0 1	0	0	0 1 1	0 0 0 0 1

NOR——按位或非指令

统一化指令

语法	操作	编译结果

nor rz, rx	RZ ← !(RZ | RX)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		nor16 rz, rx;
		else
		nor32 rz, rz, rx;

nor rz, rx, ry	RZ ← !(RX | RY)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16),
		then
		nor16 rz, rx
		else
		nor32 rz, rx, ry

说明： 将 RX 与 RY/RZ 的值按位或，然后按位取非，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← !(RZ | RX)

语法： nor16 rz, rx

说明： 将 RZ 与 RX 的值按位或，然后按位取非，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 1	RZ	RX	1 0

32位指令

操作： RZ ← !(RX | RY)

语法： nor32 rz, rx, ry

说明： 将 RX 与 RY 的值按位或，然后按位取非，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 1 0 0 1		0 0 1 0 0		RZ	

NOT——按位非指令#

统一化指令

语法	操作	编译结果

not rz	RZ ← !(RZ)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<16), then
		not16 rz;
		else
		not32 rz, rz;

not rz, rx	RZ ← !(RX)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (z<16), then
		not16 rz;
		else
		not32 rz, rx;

说明： 将 RZ/RX 的值按位取反，把结果存在 RZ。

注意，该指令是 nor rz, rz 和 nor rz, rx, rx 的伪指令。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← !(RZ)

语法： not16 rz

说明： 将 RZ 的值按位取反，把结果存在 RZ。

注意，该指令是 nor16 rz, rz 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 1	RZ	RZ	1 0

32位指令

操作： RZ ← !(RX)

语法： not32 rz, rx

说明： 将 RX 的值按位取反，把结果存在 RZ。

注意，该指令是 nor32 rz, rx, rx 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RX	RX	0 0 1 0 0 1	0 0 1 0 0	RZ

OR——按位或指令

统一化指令

语法	操作	编译结果

or rz, rx	RZ ← RZ | RX	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		or16 rz, rx ;
		else
		or32 rz, rz, rx;

or rz, rx, ry	RZ ← RX | RY	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (x<16) and (z<16), then
		or16 rz, rx
		else
		or32 rz, rx, ry

说明： 将 RX 与 RY/RZ 的值按位或，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ | RX

语法： or16 rz, rx

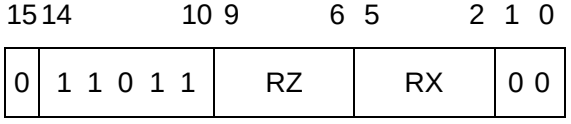
说明： 将 RZ 与 RX 的值按位或，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：



32位指令

操作： RZ ← RX | RY

语法： or rz, rx, ry

说明： 将 RX 与 RY 的值按位或，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 1 0 0 1	0 0 0 0 1	RZ

ORI——立即数按位或指令

统一化指令

语法	操作	编译结果
ori rz, rx, imm16	RZ ← RX | zero_extend(IMM16)	仅存在 32 位指令。 ori32 rz, rx, imm16

说明： 将 16 位立即数零扩展至 32 位，然后与 RX 的值进行按位或操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

32位指令

操作： RZ ← RX | zero_extend(IMM16)

语法： ori32 rz, rx, imm16

说明： 将 16 位立即数零扩展至 32 位，然后与 RX 的值进行按位或操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFFF。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	0
1	1 1 0 1 1		RZ		RX		IMM16	

POP——出栈指令

统一化指令

语法	操作	编译结果

pop reglist	从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，然后更新堆栈寄存器到堆栈存储器的顶端，并子程序返回；
	dst ← {reglist}; addr ← SP;
	foreach (reglist){
	Rdst ← MEM[addr];
	dst ← next {reglist};
	addr ← addr + 4;
	}
	sp ← addr;
	PC ← R15 & 0xfffffffe;
		pop16 reglist

说明： 从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，然后更新堆栈寄存器到堆栈存储器的顶端，并子程序返回。

```
dst ← {reglist}; addr ← SP;
foreach ( reglist ){
   Rdst ← MEM[addr];
   dst ← next {reglist};
   addr ← addr + 4;
}
sp ← addr;
PC ← R15 & 0xfffffffe;
```

语法： pop16 reglist

说明： 从堆栈存储器加载连续的多个字到一片连续的寄存器堆中，更新堆栈指针寄存器，然后实现子程序返回功能，即程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。采用堆栈指针寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r4 – r11, r15。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

15	14 10	9	8	7 6	5	4	3 0
0	0 0 1 0 1	0	0	1 0	0	R15	LIST1

LIST1 域——指定寄存器 r4-r11 是否在寄存器列表中。

0000——r4-r11 不在寄存器列表中

0001——r4 在寄存器列表中

0010——r4-r5 在寄存器列表中

0011——r4-r6 在寄存器列表中

……

1000——r4-r11 在寄存器列表中

R15 域——指定寄存器 r15 是否在寄存器列表中。

0——r15 不在寄存器列表中

1——r15 在寄存器列表中

PUSH——压栈指令

统一化指令

语法	操作	编译结果

push reglist	将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端；
	src ← {reglist}; addr ← SP;
	foreach (reglist){
	addr ← addr - 4; MEM[addr] ← Rsrc;
	src ← next {reglist};
	}
	sp ← addr;
		push16 reglist

说明： 将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 将寄存器列表中的字存储到堆栈存储器中

```
src ← {reglist}; addr ← SP;
foreach ( reglist ){
   MEM[addr] ← Rsrc;
   src ← next {reglist};
   addr ← addr - 4;
}
sp ← addr
```

语法： push16 reglist

说明： 将寄存器列表中的字存储到堆栈存储器中，然后更新堆栈寄存器到堆栈存储器的顶端。采用堆栈寄存器直接寻址方式。

影响标志位： 无影响

限制： 寄存器的范围为 r4 – r11, r15。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

15	14 10	9	8	7 6	5	4	3 0
0	0 0 1 0 1	0	0	1 1	0	R15	LIST1

LIST1 域——指定寄存器 r4-r11 是否在寄存器列表中。

0000——r4-r11 不在寄存器列表中

0001——r4 在寄存器列表中

0010——r4-r5 在寄存器列表中

0011——r4-r6 在寄存器列表中

……

1000——r4-r11 在寄存器列表中

R15 域——指定寄存器 r15 是否在寄存器列表中。

0——r15 不在寄存器列表中

1——r15 在寄存器列表中

REVB——字节倒序指令

统一化指令

语法	操作	编译结果
revb rz, rx	RZ[31:24] ← RX[7:0]; RZ[23:16] ← RX[15:8]; RZ[15:8] ← RX[23:16]; RZ[7:0] ← RX[31:24];	仅存在 16 位指令 revb16 rz, rxjin

说明： 把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ[31:24] ← RX[7:0];
RZ[23:16] ← RX[15:8];
RZ[15:8] ← RX[23:16];
RZ[7:0] ← RX[31:24];

语法： revb16 rz, rx

说明： 把 RX 的值按字节取倒序，各字节内部的位顺序保持不变，结果存入 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 1 0	RZ	RX	1 0

REVH——半字字节倒序指令

统一化指令

语法	操作	编译结果
revh rz, rx	RZ[31:24] ← RX[23:16]; RZ[23:16] ← RX[31:24]; RZ[15:8] ← RX[7:0]; RZ[7:0] ← RX[15:8];	仅存在 16 位指令。 revh16 rz, rx

说明： 把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。

影响标志位： 无影响

异常： 无

16位指令

操作：
RZ[31:24] ← RX[23:16];
RZ[23:16] ← RX[31:24];
RZ[15:8] ← RX[7:0];
RZ[7:0] ← RX[15:8];

语法： revh16 rz, rx

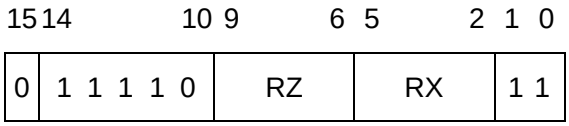
说明： 把 RX 的值在半字内按字节取倒序，即分别交换高半字内的两个字节和低半字内的两个字节，两个半字间的顺序和各字节内的位顺序保持不变，结果存入 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：



ROTL——循环左移指令

统一化指令

语法	操作	编译结果

rotl rz, rx	RZ ← RZ <<<< RX[5:0]	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		rotl16 rz, rx ;
		else
		rotl32 rz, rz, rx;

rotl rz, rx, ry	RZ ← RX <<<< RY[5:0]	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		rotl16 rz, ry
		else
		rotl32 rz, rx, ry

说明： 对于 rotl rz, rx，将 RZ 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0]的值大于 31，那么 RZ 将被清零。

对于 rotl rz, rx, ry，将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ <<<< RX[5:0]

语法： rotl16 rz, rx

说明： 将 RZ 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RX 低 6 位（RX[5:0]）的值决定；如果 RX[5:0]的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 1 0 0	RZ	RX	1 1

32位指令

操作： RZ ← RX <<<< RY[5:0]

语法： rotl32 rz, rx, ry

说明： 将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 RY 低 6 位（RY[5:0]）的值决定；如果 RY[5:0] 的值大于 31，那么 RZ 将被清零。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 1 0 0 0 0	0 1 0 0 0	RZ

ROTLI——立即数循环左移指令

统一化指令

语法	操作	编译结果
rotli rz, rx, imm5	RZ ← RX <<<< IMM5	rotli32 rz, rx, imm5;

说明： 将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

32位指令

操作： RZ ← RX <<<< IMM5

语法： rotli32 rz, rx, imm5

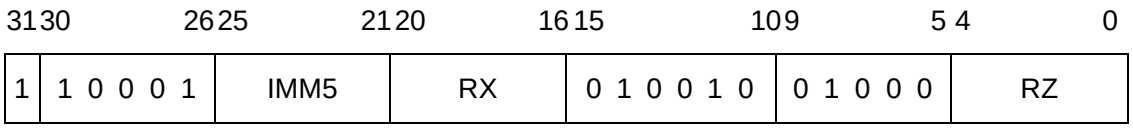
说明： 将 RX 的值进行循环左移（原值左移，右侧移入左侧移出的位），结果存入 RZ，左移位数由 5 位立即数（IMM5）的值决定；如果 IMM5 的值等于 0，那么 RZ 的值将与 RX 相同。

影响标志位： 无影响

限制： 立即数的范围为 0-31。

异常： 无

指令格式：



RSUB——反向减法指令#

统一化指令

语法	操作	编译结果
rsub rz, rx, ry	RZ ← RY - RX	仅存在 32 位指令。 rsub32 rz, rx, ry

说明： 将 RY 的值减去 RX 值，并把结果存在 RZ 中。

注意，该指令是 subu rz, ry, rx 的伪指令。

影响标志位： 无影响

异常： 无

32位指令

操作： RZ ← RY - RX

语法： rsub32 rz, rx, ry

说明： 将 RY 的值减去 RX 值，并把结果存在 RZ 中。

注意，该指令是 subu32 rz, ry, rx 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RX	RY	0 0 0 0 0 0	0 0 1 0 0	RZ

RTS——子程序返回指令#

统一化指令

语法	操作	编译结果
rts	程序跳转到链接寄存器指定的位置 PC ← R15 & 0xfffffffe	总是编译为 16 位指令。 rts16

说明： 程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS16 指令的跳转范围是全部 4GB 地址空间。

该指令用于实现子程序返回功能。

注意，该指令是 jmp r15 的伪指令。

影响标志位： 无影响

异常： 无

16位指令

操作： 程序跳转到链接寄存器指定的位置

PC ← R15 & 0xfffffffe

语法： rts16

说明： 程序跳转到链接寄存器 R15 指定的位置，链接寄存器的最低位被忽略。RTS16 指令的跳转范围是全部 4GB 地址空间。

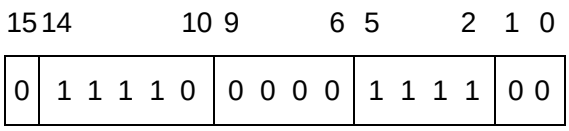
该指令用于实现子程序返回功能。

注意，该指令是 jmp16 r15 的伪指令。

影响标志位： 无影响

异常： 无

指令格式：



RTE——异常和普通中断返回指令

统一化指令

语法	操作	编译结果
rte	异常和普通中断返回 PC ← EPC，PSR ← EPSR	仅存在 32 位指令。 rte32

属性： 特权指令

说明： PC 值恢复为保存在控制寄存器 EPC 中的值，PSR 值恢复为保存在 EPSR 的值，指令执行从新的 PC 地址处开始。

影响标志位： 无影响

异常： 特权违反异常

32位指令

操作： 异常和普通中断返回
PC ← EPC，PSR ← EPSR

语法： rte32

属性： 特权指令

说明： PC 值恢复为保存在控制寄存器 EPC 中的值，PSR 值恢复为保存在 EPSR 的值，指令执行从新的 PC 地址处开始。

影响标志位： 无影响

异常： 特权违反异常

指令格式：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0

SEXTB——字节提取并有符号扩展指令#

统一化指令

语法	操作	编译结果
sextb rz, rx	RZ ← sign_extend(RX[7:0]);	仅存在 16 位指令。 sextb16 rz, rx

说明： 将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← sign_extend(RX[7:0]);

语法： sextb16 rz, rx

说明： 将 RX 的低字节（RX[7:0]）符号扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 1 0 1	RZ	RX	1 0

SEXTH——半字提取并有符号扩展指令#

统一化指令

语法	操作	编译结果
sexth rz, rx	RZ ← sign_extend(RX[15:0]);	仅存在 16 位指令 sexth16 rz, rx

说明： 将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← sign_extend(RX[15:0]);

语法： sexth16 rz, rx

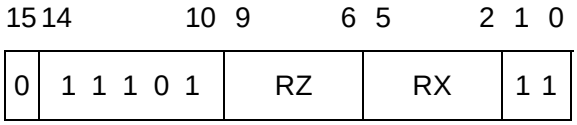
说明： 将 RX 的低半字（RX[15:0]）符号扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式



ST.B——字节存储指令

统一化指令

语法	操作	编译结果

st.b rz, (rx, disp)	将寄存器中的最低字节存储到存储器中
	MEM[RX + zero_extend(offset)] ← RZ[7:0]
		根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (disp<32) and (x<7) and (z<7), then
		st16.b rz, (rx, disp);
		else
		st32.b rz, (rx, disp);

说明： 将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.B 指令可以寻址+4KB 地址空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 将寄存器中的最低字节存储到存储器中

MEM[RX + zero_extend(offset)] ← RZ[7:0]

语法： st16.b rz, (rx, disp)

说明： 将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.B 指令可以寻址+32B 的空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

15	14 11	10 8	7 5	4 0
1	0 1 0 0	RX	RZ	IMM5

32位指令

操作： 将寄存器中的最低字节存储到存储器中

MEM[RX + zero_extend(offset)] ← RZ[7:0]

语法： st32.b rz, (rx, disp)

说明： 将寄存器 RZ 中的最低字节存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.B 指令可以寻址+4KB 地址空间。

注意，偏移量 DISP 即二进制操作数 Offset。

影响标志位： 无影响

异常： 访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

31	30 26	25 21	20 16	15 12	11 0
1	1 0 1 1 1	RZ	RX	0 0 0 0	Offset

ST.H——半字存储指令

统一化指令

语法	操作	编译结果

st.h rz, (rx, disp)	将寄存器中的最低字节存储到存储器中
	MEM[RX + zero_extend(offset<< 1)] ← RZ[15:0]
		根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (disp<64)and(x<7)and(z<7), then
		st16.h rz, (rx, disp);
		else
		st32.h rz, (rx, disp);

说明： 将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.H 指令可以寻址+8KB 地址空间。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 将寄存器中的低半字存储到存储器中

MEM[RX + zero_extend(offset << 1)] ← RZ[15:0]

语法： st16.h rz, (rx, disp)

说明： 将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.H 指令可以寻址+64B 的空间。

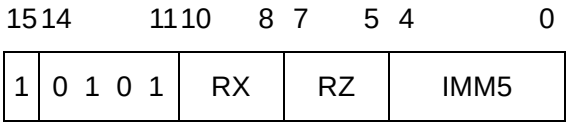
注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：



32位指令

操作： 将寄存器中的低半字存储到存储器中

MEM[RX + zero_extend(offset << 1)] ← RZ[15:0]

语法： st32.h rz, (rx, disp)

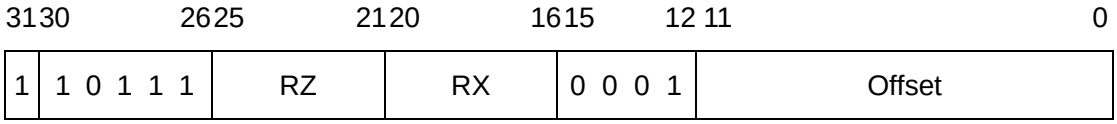
说明： 将寄存器 RZ 中的低半字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移 1 位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.H 指令可以寻址+8KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移 1 位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：



ST.W——字存储指令

统一化指令

语法	操作	编译结果

st.w rz, (rx, disp)	将寄存器中的字存储到存储器中
	MEM[RX + zero_extend(offset<< 2)] ← RZ[31:0]
		根据偏移量和寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x=sp) and (z<7) and (disp < 1024),
		st16.w rz, (sp, disp);
		else if (disp<128) and (x<7) and (z<7),
		st16.w rz, (rx, disp);
		else
		st32.w rz, (rx, disp);

说明： 将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST.W 指令可以寻址+16KB 地址空间。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

16位指令

操作： 将寄存器中的字存储到存储器中

MEM[RX + zero_extend(offset << 2)] ← RZ[31:0]

语法： st16.w rz, (rx, disp)

st16.w rz, (sp, disp)

说明： 将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。当 rx=sp 时，存储器的有效地址由基址寄存器 RX 加上左移两位的 8 位相对偏移量无符号扩展到 32 位后的值得到。当 rx 为其它寄存器时，存储器的有效地址由基址寄存器 RX 加上左移两位的 5 位相对偏移量无符号扩展到 32 位后的值得到。ST16.W 指令可以寻址+1KB 的空间。

注意，偏移量 DISP 是二进制操作数 IMM5 左移两位得到的。当基址寄存器 RX 为 SP 时，偏移量 DISP 是二进制操作数{IMM3, IMM5}左移两位得到的。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 读无效异常

指令格式：

st16.w rz, (rx, disp)

15	14 11	10 8	7 5	4 0
1	0 1 1 0	RX	RZ	IMM5

st16.w rz, (sp, disp)

15	14 11	10 8	7 5	4 0
1	0 1 1 1	IMM3	RZ	IMM5

32位指令

操作： 将寄存器中的字存储到存储器中

MEM[RX + zero_extend(offset << 2)] ← RZ[31:0]

语法： st32.w rz, (rx, disp)

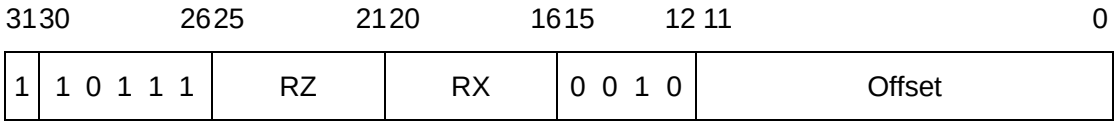
说明： 将寄存器 RZ 中的字存储到存储器中。采用寄存器加无符号立即数偏移量的寻址方式。存储器的有效地址由基址寄存器 RX 加上左移两位的 12 位相对偏移量无符号扩展到 32 位后的值得到。ST32.W 指令可以寻址+16KB 地址空间。

注意，偏移量 DISP 是二进制操作数 Offset 左移两位得到的。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：



STM——连续多字存储指令

统一化指令

语法	操作	编译结果

stm ry-rz, (rx)	将一片连续的寄存器堆中内容依次存储到一片连续的存储器地址上。
	src ← Y; addr ← RX;
	for (n = 0; n <=(Z-Y); n++){
	MEM[addr] ← Rsrc;
	src ← src + 1;
	addr ← addr + 4;
	}
		仅存在 32 位指令。
		stm32 ry-rz, (rx)

说明： 将从 RY 开始的一片连续的寄存器堆中的内容依次存储到一片连续的存储器地址上，即将寄存器 RY 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 RY+1 的内容存到存储器指定地址开始的第二个字的地址上，依次类推，将寄存器 RZ 的内容存到存储器指定地址开始的最后一个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。

影响标志位： 无影响

限制： RZ 应当大于等于 RY。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32位指令

操作： 将一片连续的寄存器堆中内容依次存储到一片连续的存储器地址上。

```
src ← Y; addr ← RX;
for (n = 0; n <= IMM5; n++){
   MEM[addr] ← Rsrc;
   src ← src + 1;
   addr ← addr + 4;
}
```

语法： stm32 ry-rz, (rx)

说明： 将从 RY 开始的一片连续的寄存器堆中的内容依次存储到一片连续的存储器地址上，即将寄存器 RY 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 RY+1 的内容存到存储器指定地址开始的第二个字的地址上，依次类推，将寄存器 RZ 的内容存到存储器指定地址开始的最后一个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。

影响标志位： 无影响

限制： RZ 应当大于等于 RY。

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 1 0 1	RY	RX	0 0 0 1 1 1	0 0 0 0 1	IMM5

IMM5 域——指定目标寄存器的个数，IMM5 = Z - Y。

00000——1 个目的寄存器

00001——2 个目的寄存器

……

11111——32 个目的寄存器

STQ——连续四字存储指令#

统一化指令

语法	操作	编译结果
stq r4-r7, (rx)	将寄存器 R4—R7 中的字依次存储到一片连续的存储器地址上。 src ← 4; addr ← RX; for (n = 0; n <= 3; n++){ MEM[addr] ← Rsrc; src ← src + 1; addr ← addr + 4; }	仅存在 32 位指令。 stq32 r4-r7, (rx);

说明： 将寄存器堆[R4,R7]（包括边界）中的字依次存储到一片连续的存储器地址上，即将寄存器 R4 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 R5 的内容存到存储器指定地址开始的第二个字的地址上，寄存器 R6 的内容存到存储器指定地址开始的第三个字的地址上，寄存器 R7 的内容存到存储器指定地址开始的第四个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。注意，该指令是 stm r4-r7, (rx)的伪指令。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

32位指令

操作： 将寄存器 R4—R7 中的字依次存储到一片连续的存储器地址上。

```
src ← 4; addr ← RX;
for (n = 0; n <= 3; n++){
   MEM[addr] ← Rsrc;
   src ← src + 1;
   addr ← addr + 4; }
```

语法： stq32 r4-r7, (rx)

说明： 将寄存器堆[R4,R7]（包括边界）中的字依次存储到一片连续的存储器地址上，即将寄存器 R4 的内容存到存储器指定地址开始的第一个字的地址上，寄存器 R5 的内容存到存储器指定地址开始的第二个字的地址上，寄存器 R6 的内容存到存储器指定地址开始的第三个字的地址上，寄存器 R7 的内容存到存储器指定地址开始的第四个字的地址上。存储器的有效地址由基址寄存器 RX 的内容决定。注意，该指令是 stm r4-r7, (rx)的伪指令。

影响标志位： 无影响

异常： 未对齐访问异常、访问错误异常、TLB 不可恢复异常、TLB 失配异常、TLB 写无效异常

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 1 0 1	0 0 1 0 0	RX	0 0 0 1 1 1	0 0 0 0 1	0 0 0 1 1

STOP——进入低功耗暂停模式指令

统一化指令

语法	操作	编译结果
stop	进入低功耗暂停模式	仅存在 32 位指令。 stop32

说明： 此指令使处理器进入低功耗模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，大部分外围设备也被停止。

影响标志位： 无影响

异常： 特权违反异常

32位指令

操作： 进入低功耗暂停模式

语法： stop32

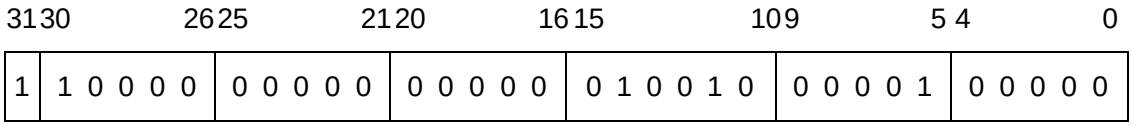
属性： 特权指令

说明： 此指令使处理器进入低功耗模式，并等待一个中断来退出这个模式。此时，CPU 时钟停止，大部分外围设备也被停止。

影响标志位： 无影响

异常： 特权违反异常

指令格式：



SUBC——无符号带借位减法指令

统一化指令

语法	操作	编译结果

subc rz, rx	RZ ← RZ - RX -（!C），
	C ← 借位
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		subc16 rz, rx;
		else
		subc32 rz, rz, rx;

subc rz, rx, ry	RZ ← RX - RY -（!C），
	C ← 借位
		根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x==z) and (y<16) and (z<16), then
		subc16 rz, ry;
		else
		subc32 rz, rx, ry;

说明： 对于 subc rz, rx，将 RZ 的值减去寄存器 RX 的值和 C 位的非值；对于 subc rz, rx, ry，将 RX 的值减去寄存器 RY 的值和 C 位的非值。把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。

影响标志位： C ← 借位

异常： 无

16位指令

操作： RZ ← RZ - RX -（!C），C ← 借位

语法： subc16 rz, rx

说明： 将 RZ 的值减去寄存器 RX 的值和 C 位的非值，并把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。

影响标志位： C ← 借位

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14	10	9	6	5	2	1	0
0	1 1 0 0 0		RZ		RX		1	1

32位指令

操作： RZ ← RX - RY -（!C），C ← 借位

语法： subc32 rz, rx, ry

说明： 将 RX 的值减去寄存器 RY 的值和 C 位的非值，并把结果存在 RZ，借位存在 C 位。对于该减法指令来说，如果发生借位，将清 C 位，反之置 C 位。

影响标志位： C ← 借位

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 0 0 0 0		0 1 0 0 0		RZ	

SUBI——无符号立即数减法指令

统一化指令

语法	操作	编译结果

subi rz, oimm12	RZ ← RZ - zero_extend(OIMM12)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (oimm12<257) and (z<8), then
		subi16 rz, oimm8;
		else
		subi32 rz, rz, oimm12;

subi rz, rx, oimm12	RZ ← RX - zero_extend(OIMM12)	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (oimm12<8) and (z<8) and (x<8), then
		subi16 rz, rx, oimm3;
		elsif (x==z) and (z<8) and (oimm12<257), then
		subi16 rz, oimm8;
		else
		subi32 rz, rx, oimm12;

说明： 将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后用 RZ/RX 的值减去该 32 位数，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x1-0x1000。

异常： 无

16位指令----1

操作： RZ ← RZ - zero_extend(OIMM8)

语法： subi16 rz, oimm8

说明： 将带偏置 1 的 8 位立即数（OIMM8）零扩展至 32 位，然后用 RZ 的值减去该 32 位数，把结果存入 RZ。

注意：二进制操作数 IMM8 等于 OIMM8 - 1。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 1-256。

异常： 无

指令格式：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	RZ			IMM8							

IMM8 域——指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM8 比起二进制操作数 IMM8 需偏置 1。

00000000——减 1

00000001——减 2

……

11111111——减 256

16位指令----2

操作： RZ ← RX - zero_extend(OIMM3)

语法： subi16 rz, rx, oimm3

说明： 将带偏置 1 的 3 位立即数（OIMM3）零扩展至 32 位，然后用 RX 的值减去该 32 位数，把结果存入 RZ。

注意：二进制操作数 IMM3 等于 OIMM3 - 1。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7；立即数的范围为 1-8。

异常： 无

指令格式：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	RX			RZ			IMM3			1	1

IMM3 域——指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM3 比起二进制操作数 IMM3 需偏置 1。

000——减 1

001——减 2

……

111——减 8

32位指令

操作： RZ ← RX - zero_extend(OIMM12)

语法： subi32 rz, rx, oimm12

说明： 将带偏置 1 的 12 位立即数（OIMM12）零扩展至 32 位，然后用 RX 的值减去该 32 位数，把结果存入 RZ。

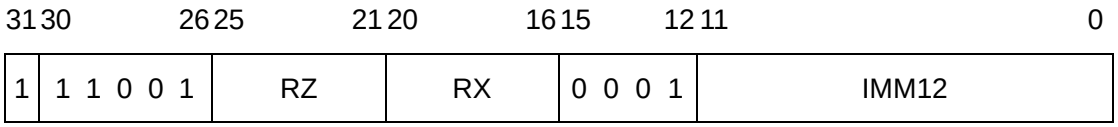
注意：二进制操作数 IMM12 等于 OIMM12 - 1。

影响标志位： 无影响

限制： 立即数的范围为 0x1-0x1000。

异常： 无

指令格式：



IMM12 域——指定不带偏置立即数的值。

注意：寄存器减去的值 OIMM12 比起二进制操作数 IMM12 需偏置 1。

000000000000——减 0x1

000000000001——减 0x2

……

111111111111——减 0x1000

SUBI(SP)——无符号（堆栈指针）立即数减法指令

统一化指令

语法	操作	编译结果
subi sp, sp, imm	SP ← SP- zero_extend(IMM)	仅存在 16 位指令。 subi sp, sp, imm

说明： 将立即数（IMM）零扩展至 32 位并左移 2 位，然后与堆栈指针（SP）的值相减，把结果存入 SP。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0x1fc。

异常： 无

16位指令

操作： SP ← SP - zero_extend(IMM)

语法： subi sp, sp, imm

说明： 将立即数（IMM）零扩展至 32 位并左移 2 位，然后与堆栈指针（SP）的值相减，把结果存入堆栈指针。

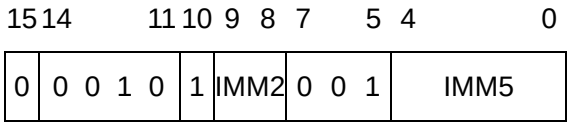
注意：立即数（IMM）等于二进制操作数{IMM2, IMM5} << 2。

影响标志位： 无影响

限制： 源与目的寄存器均为堆栈指令寄存器（R14）；立即数的范围为 (0x0-0x7f) << 2。

异常： 无

指令格式：



IMM 域——指定不带移位的立即数的值。

注意：加到寄存器里的值 IMM 比起二进制操作数{IMM2, IMM5}需左移 2 位。

{00, 00000}——减 0x0

{00, 00001}——减 0x4

……

{11, 11111}——减 0x1fc

SUBU——无符号减法指令

统一化指令

语法	操作	编译结果

subu rz, rx
sub rz, rx
	RZ ← RZ - RX	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<16) and (x<16), then
		subu16 rz, rx;
		else
		subu32 rz, rz, rx;

subu rz, rx, ry	RZ ← RX - RY	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (z<8) and (x<8) and (y<8), then
		subu16 rz, rx, ry;
		elsif (x==z) and (z<16) and (y<16), then
		subu16 rz, ry;
		else
		subu32 rz, rx, ry;

说明： 对于 subu rz, rx，将 RZ 的值减去 RX 值，并把结果存在 RZ 中。

对于 subu rz, rx, ry，将 RX 的值减去 RY 值，并把结果存在 RZ 中。

影响标志位： 无影响

异常： 无

16位指令----1

操作： RZ ← RZ - RX

语法： subu16 rz, rx

sub16 rz, rx

说明： 将 RZ 的值减去 RX 值，并把结果存在 RZ 中。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 0 0	RZ	RX	1 0

16位指令----2

操作： RZ ← RX- RY

语法： subu16 rz, rx, ry

sub16 rz, rx, ry

说明： 将 RX 的值减去 RY 值，并把结果存在 RZ 中。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r7。

异常： 无

指令格式：

15	14 11	10 8	7 5	4 2	1 0
0	1 0 1 1	RX	RZ	RY	0 1

32位指令

操作： RZ ← RX - RY

语法： subu32 rz, rx, ry

说明： 将 RX 的值减去 RY 值，并把结果存在 RZ 中。

影响标志位： 无影响

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	RY	RX	0 0 0 0 0 0	0 0 1 0 0	RZ

SYNC——CPU 同步指令

统一化指令

语法	操作	编译结果
sync	使 CPU 同步	仅存在 32 位指令。 sync32

说明： 当处理器碰到 sync 指令时，指令就会被悬挂起来直到所有外面的操作全都完成，即没有未完成的指令。

影响标志位： 无影响

异常： 无

32位指令

操作： 使 CPU 同步

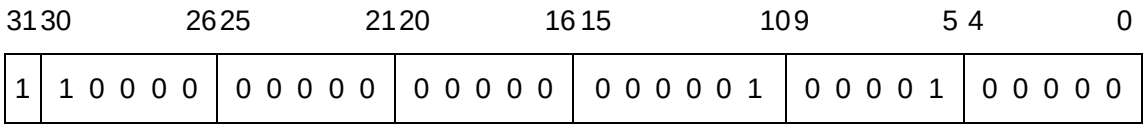
语法： sync32

说明： 当处理器碰到 sync 指令时，指令就会被悬挂起来直到所有外面的操作全都完成，即没有未完成的指令。

影响标志位： 无影响

异常： 无

指令格式：



TRAP——操作系统陷阱指令

统一化指令

语法	操作	说明
trap 0, trap 1 trap 2, trap 3	引起陷阱异常发生	仅存在 32 位指令。 trap32 0, trap32 1 trap32 2, trap32 3

说明： 当处理器碰到 trap 指令时，发生陷阱异常操作。

影响标志位： 无影响

异常： 陷阱异常

32位指令

操作： 引起陷阱异常发生

语法： trap32 0,
trap32 1,
trap32 2,
trap32 3

说明： 当处理器碰到 trap 指令时，发生陷阱异常操作。

影响标志位： 无影响

异常： 陷阱异常

指令格式：

trap32 0

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 1 0 0 0		0 0 0 0 1		0 0 0 0 0	

trap32 1

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 1 0 0 1		0 0 0 0 1		0 0 0 0 0	

trap32 2

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 1 0 1 0		0 0 0 0 1		0 0 0 0 0	

trap32 3

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 0		0 0 0 0 0		0 0 0 0 0		0 0 1 0 1 1		0 0 0 0 1		0 0 0 0 0	

TST——零测试指令

统一化指令

语法	操作	编译结果

tst rx, ry	If (RX & RY) != 0, then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令。
		tst16 rx, ry

说明： 测试 RX 和 RY 的值按位与的结果。

如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据按位与结果设置条件位 C

异常： 无

16位指令

操作： If (RX & RY) != 0, then

C ← 1;

else

C ← 0;

语法： tst16 rx, ry

说明： 测试 RX 和 RY 的值按位与的结果。

如果结果不等于 0，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据按位与结果设置条件位 C

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 0	RY	RX	1 0

TSTNBZ——无字节等于零寄存器测试指令

统一化指令

语法	操作	编译结果

tstnbz16 rx	If ((RX[31:24] != 0)
	&(RX[23:16] != 0)
	&(RX[15: 8] != 0)
	&(RX[7 : 0] != 0)), then
	C ← 1;
	else
	C ← 0;
		仅存在 16 位指令。
		tstnbz16 rx

说明： 测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据按位与结果设置条件位 C

异常： 无

16位指令

操作：

```
If  ( (RX[31:24] != 0)
    &(RX[23:16] != 0)
    &(RX[15: 8 ] != 0)
    &(RX[ 7 : 0 ] != 0) ), then
  C ← 1;
else
  C ← 0;
```

语法： tstnbz16 rx

说明： 测试 RX 中是否没有字节等于零。如果 RX 没有字节等于零，则设置条件位 C；否则，清除条件位 C。

影响标志位： 根据按位与结果设置条件位 C

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 0	0 0 0 0	RX	1 1

WAIT——进入低功耗等待模式指令

统一化指令

语法	操作	编译结果
wait	进入低功耗等待模式	仅存在 32 位指令。 wait32

属性： 特权指令

说明： 此指令停止当前指令执行，并等待一个中断，此时 CPU 时钟停止。所有的外围设备都仍在继续运行，并有可能会产生中断而引起 CPU 从等待模式退出。

影响标志位： 无影响

异常： 特权违反指令

32位指令

操作： 进入低功耗等待模式

语法： wait32

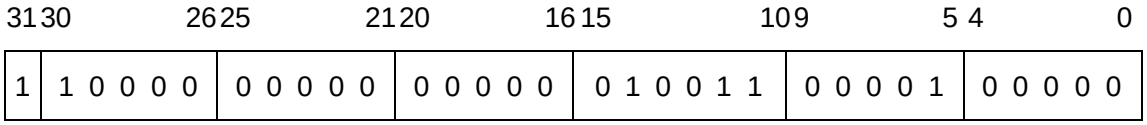
属性： 特权指令

说明： 此指令停止当前指令执行，并等待一个中断，此时 CPU 时钟停止。所有的外围设备都仍在继续运行，并有可能会产生中断而引起 CPU 从等待模式退出。

影响标志位： 无影响

异常： 特权违反指令

指令格式：



XOR——按位异或指令

统一化指令

语法	操作	编译结果

xor rz, rx	RZ ← RZ ^ RX	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (x<16) and (z<16), then
		xor16 rz, rx;
		else
		xor32 rz, rz, rx;

xor rz, rx, ry	RZ ← RX ^ RY	根据寄存器的范围编译为对应的 16 位或 32 位指令。
		if (y==z) and (z<16) and (x<16), then
		xor16 rz, rx;
		else
		xor32 rz, rx, ry;

说明： 将 RX 与 RZ/RY 的值按位异或，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← RZ ^ RX

语法： xor16 rz, rx

说明： 将 RZ 与 RX 的值按位异或，并把结果存在 RZ。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15	14 10	9 6	5 2	1 0
0	1 1 0 1 1	RZ	RX	0 1

32位指令

操作： RZ ← RX ^ RY

语法： xor32 rz, rx, ry

说明： 将 RX 与 RY 的值按位异或，并把结果存在 RZ。

影响标志位： 无影响

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		RY		RX		0 0 1 0 0 1		0 0 0 1 0		RZ	

XORI——立即数按位异或指令

统一化指令

语法	操作	编译结果
xori rz, rx, imm16	RZ ← RX ^ zero_extend(IMM12)	仅存在 32 位指令。 xori32 rz, rx, imm12

说明： 将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位异或操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFF。

异常： 无

32位指令

操作： RZ ← RX ^ zero_extend(IMM12)

语法： xori32 rz, rx, imm12

说明： 将 12 位立即数零扩展至 32 位，然后与 RX 的值进行按位异或操作，把结果存入 RZ。

影响标志位： 无影响

限制： 立即数的范围为 0x0-0xFFF。

异常： 无

指令格式：

31	30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
1	1 1 0 0 1	RZ	RX	0 1 0 0	IMM12

XSR——扩展右移指令

统一化指令

语法	操作	编译结果
xsr rz, rx, oimm5	{RZ,C} ←{RX,C} >>>> OIMM5	仅存在 32 位指令。 xsr32 rz, rx, oimm5

说明： 将 RX 带条件位 C 的值（{RX,C}）进行循环右移（原值右移，左侧移入右侧移出的位），把移位结果的最低位（[0]）存入条件位 C，高位（[32:1]）存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位。

影响标志位： C ← RX[OIMM5 - 1]

限制： 立即数的范围为 1-32。

异常： 无

32位指令

操作： {RZ,C} ←{RX,C} >>>> OIMM5

语法： xsr32 rz, rx, oimm5

说明： 将 RX 带条件位 C 的值（{RX,C}）进行循环右移（原值右移，左侧移入右侧移出的位），把移位结果的最低位（[0]）存入条件位 C，高位（[32:1]）存入 RZ，右移位数由带偏置 1 的 5 位立即数（OIMM5）的值决定。如果 OIMM5 的值等于 32，那么条件位 C 为 RX 的最高位。

注意：二进制操作数 IMM5 等于 OIMM5 - 1。

影响标志位： C ← RX[OIMM5 - 1]

限制： 立即数的范围为 1-32。

异常： 无

指令格式：

31	30 26	25 21	20 16	15 10	9 5	4 0
1	1 0 0 0 1	IMM5	RX	0 1 0 0 1 1	0 1 0 0 0	RZ

IMM5 域——指定不带偏置立即数的值。

注意：移位的值 OIMM5 比起二进制操作数 IMM5 需偏置 1。

00000——移 1 位

00001——移 2 位

……

11111——移 32 位

XTRB0——提取字节 0 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb0 rz, rx	RZ ← zero_extend(RX[31:24]);
	if (RX[31:24] == 0), then
	C ← 0;
	else
	C ← 1;
		仅存在 32 位指令。
		xtrb0.32 rz, rx

说明： 提取 RX 的字节 0（RX[31:24]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

32位指令

操作：
RZ ← zero_extend(RX[31:24]);
if (RX[31:24] == 0), then
C ← 0;
else
C ← 1;

语法： xtrb0.32 rz, rx

说明： 提取 RX 的字节 0（RX[31:24]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

指令格式：

31 30	26 25	21 20	16 15	10 9	5 4	0

1	1 0 0 0 1	0 0 0 0 0	RX	0 1 1 1 0 0	0 0 0 0 1	RZ

XTRB1——提取字节 1 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb1 rz, rx	RZ ← zero_extend(RX[23:16]);
	if (RX[23:16] == 0), then
	C ← 0;
	else
	C ← 1;
		仅存在 32 位指令。
		xtrb1.32 rz, rx

说明： 提取 RX 的字节 1（RX[23:16]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

32位指令

操作：

```
RZ ← zero_extend(RX[23:16]);
if (RX[23:16] == 0), then
  C ← 0;
else
  C ← 1;
```

语法： xtrb1.32 rz, rx

说明： 提取 RX 的字节 1（RX[23:16]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		0 0 0 0 0		RX		0 1 1 1 0 0		0 0 0 1 0		RZ	

XTRB2——提取字节 2 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb2 rz, rx	RZ ← zero_extend(RX[15:8]);
	if (RX[15:8] == 0), then
	C ← 0;
	else
	C ← 1;
		仅存在 32 位指令。
		xtrb2.32 rz, rx

说明： 提取 RX 的字节 2（RX[15:8]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

32位指令

操作：

```
RZ ← zero_extend(RX[15:8]);
if (RX[15:8] == 0), then
  C ← 0;
else
  C ← 1;
```

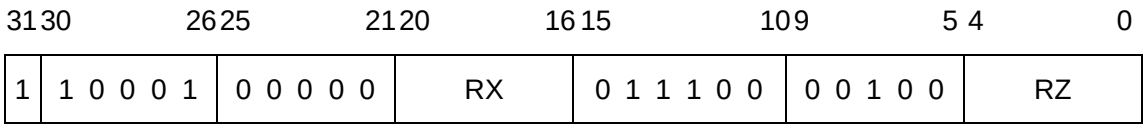
语法： xtrb2.32 rz, rx

说明： 提取 RX 的字节 2（RX[15:8]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

指令格式：



XTRB3——提取字节 3 并无符号扩展指令

统一化指令

语法	操作	编译结果

xtrb3 rz, rx	RZ ← zero_extend(RX[7:0]);
	if (RX[7:0] == 0), then
	C ← 0;
	else
	C ← 1;
		仅存在 32 位指令。
		xtrb3.32 rz, rx

说明： 提取 RX 的字节 3（RX[7:0]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

32位指令

操作：

```
RZ ← zero_extend(RX[7:0]);
if (RX[7:0] == 0), then
  C ← 0;
else
  C ← 1;
```

语法： xtrb3.32 rz, rx

说明： 提取 RX 的字节 3（RX[7:0]）到 RZ 的低位（RZ[7:0]），并进行零扩展。如果结果等于 0，则清除 C 位，反之设置 C 位。

影响标志位： 如果结果等于 0，则清除 C 位，反之设置 C 位。

异常： 无

指令格式：

31	30	26	25	21	20	16	15	10	9	5	4	0
1	1 0 0 0 1		0 0 0 0 0		RX		0 1 1 1 0 0		0 1 0 0 0		RZ	

ZEXTB——字节提取并无符号扩展指令#

统一化指令

语法	操作	编译结果
zextb rz, rx	RZ ← zero_extend(RX[7:0]);	仅存在 16 位指令。 zextb16 rz, rx;

说明： 将 RX 的低字节（RX[7:0]）零扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← zero_extend(RX[7:0]);

语法： zextb16 rz, rx

说明： 将 RX 的低字节（RX[7:0]）零扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式：

15 14 10 9 6 5 2 1 0

0	1 1 1 0 1	RZ	RX	0 0

ZEXTH——半字提取并无符号扩展指令#

统一化指令

语法	操作	编译结果
zexth rz, rx	RZ ← zero_extend(RX[15:0]);	仅存在 16 位指令。 zexth16 rz, rx

说明： 将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

异常： 无

16位指令

操作： RZ ← zero_extend(RX[15:0]);

语法： zexth16 rz, rx

说明： 将 RX 的低半字（RX[15:0]）零扩展至 32 位，结果存在 RZ 中。

影响标志位： 无影响

限制： 寄存器的范围为 r0-r15。

异常： 无

指令格式

15 14 10 9 6 5 2 1 0

0	1 1 1 0 1	RZ	RX	0 1

